Defence Research and Development Canada   Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

A Navigation and Decision Making Architecture for Unmanned Ground Vehicles

Implementation and Results with the Raptor UGV

J. Giesbrecht, J. Collier, G. Broten, S. Monckton, and D. Mackay

Defence R&D Canada

Technical Report

DRDC Suffield TR 2007-300

December 2007

Canada

A Navigation and Decision Making Architecture for Unmanned Ground Vehicles

Implementation and Results with the Raptor UGV

J. Giesbrecht, J. Collier, G. Broten, S. Monckton, and D. Mackay
Defence R&D Canada – Suffield

Defence R&D Canada – Suffield

Technical Report

DRDC Suffield TR 2007-300

December 2007

Principal Author

Original signed by J. Giesbrecht

J. Giesbrecht

Approved by

Original signed by D.M. Hanna

D. M. Hanna
Head/AISS

Approved for release by

Original signed by Dr. P.A. D'Agostino

Dr. P. A. D'Agostino
Head/Document Review Panel

Abstract

Researchers at Defence R&D Canada – Suffield, under the Autonomous Land Systems (ALS) and Cohort projects, have been working to extend/enhance the capabilities of Unmanned Ground Vehicles (UGVs) beyond tele-operation. The goal is to create robotic platforms that are effective with minimal human supervision in outdoor environments. This report is a summary of the progress made in high level vehicle control, specifically the implementation and testing of algorithms providing point-to-point navigation and decision making capabilities for UGVs. To reach goals by traversing unknown terrain requires a number of navigation functions, including path tracking, obstacle avoidance, path planning and decision making modules. This report presents details of the theoretical underpinnings, the software design architecture, and results of implementing autonomous navigation and decision making software on a robotic platform, given competing priorities and limited sensing technologies.

Résumé

Les chercheurs de R & D pour la défense Canada – Suffield ont travaillé, dans le cadre des projets de Systèmes terrestres autonomes (STA) et Cohorte, à étendre et améliorer les capacités des véhicules terrestres sans pilote au-delà de la commande à distance. Le but est de créer des plates-formes robotiques ayant un minimum de supervision humaine qui soient efficaces en milieux extérieurs. Ce rapport fait le sommaire des progrès réalisés sur des véhicules autonomes sans pilote ayant des commandes de haut niveau, surtout dans le domaine de l'implémentation et des essais d'algorithmes permettant la navigation de point à point et des capacités de prise de décision. Être en mesure d'atteindre des buts, en traversant un terrain inconnu, requiert un certain nombre de fonctions de navigation, dont le suivi de parcours, l'évitement d'obstacles, la planification de parcours et des modules de prises de décision. Ce rapport présente des détails concernant les stratégies sous-jacentes, l'architecture de la conception des logiciels et les résultats de l'implémentation de la navigation autonome et du logiciel de prise de décision sur une plate-forme robotique, en tenant compte des priorités concurrentes et des limitations des technologies de détection.

This page intentionally left blank.

Executive summary

A Navigation and Decision Making Architecture for Unmanned Ground Vehicles

J. Giesbrecht, J. Collier, G. Broten, S. Monckton, D. Mackay; DRDC Suffield
TR 2007-300; Defence R&D Canada – Suffield; December 2007.

Background: This document focuses on four key tasks to the navigation of Unmanned Ground Vehicles (UGVs): path tracking, local navigation, global path planning, and decision making or behaviour arbitration. Path tracking enables a vehicle to follow a set of waypoints designated by a human user or another autonomous agent. Local navigation uses robot sensors to avoid obstacles and find safe routes to travel over terrain. Global path planning uses map data provided by a user beforehand, or accumulated in the robot's travel, to find an optimal, safe route to a long term goal. Finally, decision making combines the output of a number of different algorithms with competing priorities into coherent, intelligent control for the UGV.

Principal Results: Researchers at Defence R&D Canada – Suffield have implemented and tested a number of navigation algorithms on the Raptor UGV. This combined navigation software achieved many practical behaviours, such as autonomous patrols of large areas, avoidance of static obstacles (buildings, vehicles, posts), avoidance of dynamic obstacles (moving people), road following, and leader following.

The algorithms used to achieve these results span across a broad range of robot control methodologies, from reactive to deliberative, including:

1. Pure Pursuit (Path Tracking) - A method for an autonomous UGV to follow a path along user defined geographic waypoints.
2. Laser Safety (Obstacle Detection) - Uses a scanning laser rangefinder to detect positive and negative obstacles and halt the vehicle if necessary. This is suitable for use with both tele-operated and autonomous UGVs.
3. Vector Field Histogram (Reactive 2-D Obstacle Avoidance) - A fast and highly reactive algorithm for dodging both stationary and moving discrete obstacles using a minimal amount of hardware and software complexity.
4. Reactive 3-D Obstacle Avoidance - A complete obstacle avoidance method, which deals with both positive and negative obstacles, as well as terrain slope and roughness. This requires more hardware and software complexity on the robot platform.
5. D* Lite (Deliberative Path Planning) - A graph search algorithm which makes it possible for a UGV to continuously determine the best path to a goal using *a priori* map data and/or onboard sensor data.
6. Arc Arbiter (Decision Making) - A method to combine the output of the above algorithms in a coherent way for vehicle control.

Significance of Results: These results indicate the usefulness of several specific UGV navigation algorithms, and their practical implementation has been demonstrated. Finally, the system has also demonstrated a method for constructing a complete UGV navigation system using a series of algorithm-based behaviours.

Sommaire

A Navigation and Decision Making Architecture for Unmanned Ground Vehicles

J. Giesbrecht, J. Collier, G. Broten, S. Monckton, D. Mackay ; DRDC Suffield TR 2007-300 ; R & D pour la défense Canada – Suffield ; décembre 2007.

Contexte : Ce document est axé sur quatre tâches clés de la navigation des véhicules autonomes sans pilote : le suivi de parcours, la navigation locale, la planification globale de parcours et l'arbitrage des prises de décision ou des comportements. Le suivi de parcours permet au véhicule de suivre un ensemble de points de cheminement désignés par un utilisateur humain ou un autre agent autonome. La navigation locale utilise les capteurs du robot pour éviter les obstacles et trouver des trajets sécuritaires pour traverser des terrains. La planification globale de parcours utilise les données cartographiques fournies auparavant par un utilisateur ou accumulées au cours du voyage du robot, pour trouver un trajet sécuritaire optimal vers un but à long terme. La prise de décision combine enfin les résultats d'un certain nombre d'algorithmes différents, ayant des priorits concurrentes, en une commande cohérente et intelligente pour le véhicule autonome.

Résultats principaux : Les chercheurs de R & D pour la défense Canada – Suffield ont implémenté et testé un certain nombre d'algorithmes de navigation sur le véhicule autonome sans pilote Raptor. Cette combinaison de logiciels de navigation a réalisé beaucoup de comportements pratiques tels que des patrouilles autonomes de grandes zones, l'évitement d'obstacles statiques (bâtiments, véhicules, piliers), l'évitement d'obstacles en mouvement (gens en déplacement), suivi sur une route et suivi d'un meneur.

Les algorithmes utilisés pour réaliser ces résultats s'étendent sur un large éventail de méthodologies de commandes robotiques, des réactives aux délibératives comprenant :

1. La poursuite pure (suivi de parcours) une méthode consistant à ce qu'un véhicule autonome suive un parcours le long de points de cheminements géographiques définis par un utilisateur.
2. La protection laser (détection d'obstacles) utilise un télémètre à laser à balayage pour détecter des obstacles positifs et négatifs et arrêter le véhicule si besoin. Ceci est utilisable avec les véhicules télé-opérés et les véhicules autonomes sans pilote.
3. L'histogramme à champ vectoriel (évitement réactif d'obstacles à 2-D) un algorithme trés réactif pour esquiver les obstacles stationnaires tout comme ceux en déplacement et pour déplacer les obstacles discrets en utilisant des appareils et des logiciels d'une complexité minimale.
4. L'évitement réactif d'obstacles 3-D une méthode d'évitement complet d'obstacles qui traite des obstacles positifs comme négatifs ainsi que de l'inclinaison et de la rugosité du terrain. Ceci requiert des appareils et des logiciels d'une plus grande complexité sur la plate-forme robotique.

5. D* Lite (Planification de parcours délibérative) un algorithme de recherche par graphes qui permet à un véhicule sans pilote de déterminer continuellement le meilleur parcours vers un but en utilisant des données cartographiques à priori et/ou des données provenant des capteurs de bord.
6. Arbitre en arc (prise de décision une méthode qui combine les résultats des algorithmes ci-dessus de manière cohérente pour la commande du véhicule.

Portée des résultats : Ces résultats indiquent l'utilité de plusieurs algorithmes de navigation spécifiques aux véhicules autonomes sans pilote et démontrent leur implémentation pratique. Le système démontre enfin une méthode de construction, d'un système complet de navigation pour véhicules autonomes sans pilote, qui utilise une série de comportements à base d'algorithmes.

Table of contents

List of figures

This page intentionally left blank.

1 Introduction

Researchers at Defence R&D Canada – Suffield, under the Autonomous Land Systems (ALS) and Cohort projects, have been working to extend/enhance the capabilities of Unmanned Ground Vehicles (UGVs) beyond tele-operation. The goal is creation of robotic platforms that are effective in outdoor environments with minimal human supervision . In pursuit of this goal, DRDC staff have investigated and implemented algorithms in a number of areas including robot sensing, perception, control, and multi-robot cooperation. This report details progress made in high level vehicle control, specifically the results of testing the algorithms providing point-to-point navigation and decision making capabilities implemented on the Raptor UGVs.

To be effective, an autonomous UGV must be able to reach goals despite obstacles and unknown terrain, while exploiting previously acquired information. This requires high level path planning, path tracking, local navigation (obstacle avoidance), and decision making (arbitration) capabilities. This report details the theoretical underpinnings, the software design architecture, and the implementation of autonomous navigation and decision making software on a robotic platform given competing priorities and limited sensing technologies. Specifically, the following algorithms have been implemented and tested at Defence R&D Canada – Suffield:

1. Path tracking (Pure Pursuit) - A method for an autonomous UGV to follow a path along user defined geographic waypoints.
2. Obstacle detection (Laser Safety) - Uses a scanning laser rangefinder to detect positive and negative obstacles and halt the vehicle if necessary. This is suitable for use with both tele-operated and autonomous UGVs.
3. Reactive 2-D obstacle avoidance (Vector Field Histogram (VFH)) - A fast and highly reactive algorithm for avoiding both stationary and moving discrete obstacles using a minimal amount of hardware and software complexity.
4. Reactive 3-D obstacle avoidance - A complete obstacle avoidance method, which deals with both positive and negative obstacles, as well as terrain slope and roughness. This algorithm requires more hardware and software complexity than the VFH algorithm.
5. Deliberative path planning (D* Lite) - A graph search algorithm which makes it possible for a UGV to continuously determine the best path to a goal using *a priori* map data and/or onboard sensor data.
6. Decision making (Arc Arbiter) - A method to combine the output of the above algorithms in a coherent way to provide vehicle control.

These navigation systems require sensing and world representation modules as input, described here in sufficient detail for reader understanding. Additionally, this report touches on the DRDC Architecture for Autonomy (AFA) [1], a set of computing middleware tools which simplify the combination of a variety of robot algorithms into a cohesive, intelligent system, operating over a TCP/IP network. This architecture was used to implement the navigation and decision making algorithms presented here.

The combined navigation software was tested on the Raptor UGV platforms shown in Figure 1. Many behaviours were achieved, such as autonomous patrols, avoidance of static obstacles (buildings, vehicles, posts), avoidance of dynamic obstacles (moving people), road following, and leader following. Results from demonstrations conducted in 2005 and 2006 are presented.

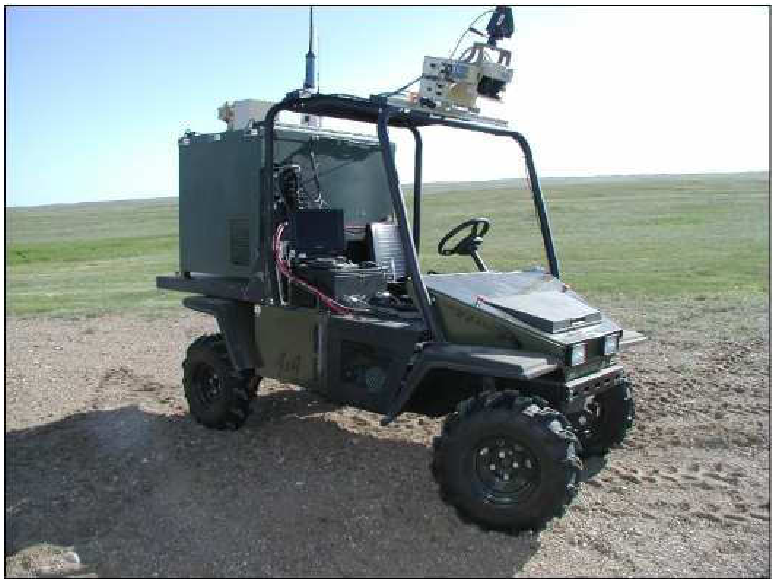


Figure 1: *One of two Koyker Raptors used in the DRDC demonstrations. Each Raptor used one or more roof mounted SICK laser scanners, Digiclops stereo cameras, DGPS positioning, 3DMG inertial measurement units, and wireless mesh networking.*

2 Background

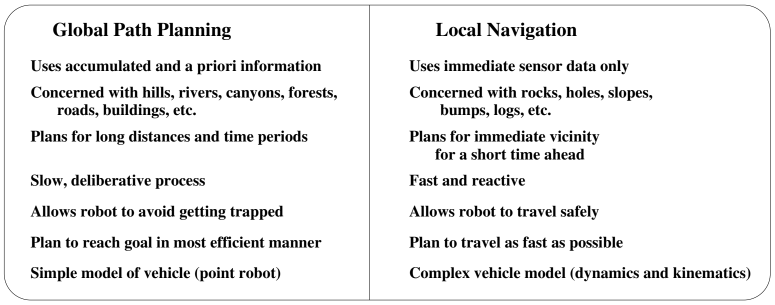
As a context for this work, a simplified UGV mission scenario is presented: a user inputs a goal position or a set of geographic way-points into the robot controller defining a path for the UGV to follow. The user may also give the robot a map of the area, possibly incomplete or inaccurate, as well as a suite of constraints for the robot, such as maintaining radio or visual contact with the user or another unit, stealth, out of bounds areas, maximum fuel consumption, etc. The robot's environment may be hazardous, contain impassable obstacles and cul-de-sacs, be dynamic in nature, and contain unpredictable elements such as humans and other robotic vehicles. Given these requirements and a set of sensors to view its environment, the robot controller must plan a path to the goal and then safely navigate there. The robot will need to re-plan as it encounters new information, and react to obstacles as they are encountered. This is a very complex process, requiring the robot to accomplish numerous simultaneous tasks:

- Sensing - Viewing the world.
- Perception - Interpreting what it sees.
- Localization - Tracking the UGV's position.
- Local navigation (Obstacle Avoidance) - Making sure the robot doesn't tip, drive into holes or bump into obstacles.
- Global path planning - Finding an optimal (in some sense: fastest, safest, etc.) way to get from the start to the goal.
- Decision making - Choosing from competing priorities to select a course of action.
- Path tracking - Following a prescribed path given to the UGV by the user or decided upon by the autonomous system.

The discussion in this document focuses on the last four tasks, which are central to the navigation task: Path Tracking, Local Navigation, Global Path Planning and Decision Making.

It is a major challenge to program robotic vehicles with the ability to find their way intelligently through a wide variety of terrains. Robust, intelligent systems for outdoor navigation arguably must use both global path planning and local navigation [2, 3]. The local navigator is a reactive process that relies on the latest sensor data to maintain vehicle safety and stability while attempting to maintain a target speed. Global path planning, is a more deliberative process that uses both *a priori* information about the world (usually in the form of a map) and information acquired from onboard sensors to provide a safe path for the robot. Because the global planner has access to a world model and the previous path that the UGV has taken, the robot is far less likely to be trapped in a cul-de-sac and the path the robot takes to reach the goal can be optimized in some sense (distance, time, fuel, etc.). The computational cost of planning with a world model can be very high. The task can be made more tractable by increasing the scale of the planning space to the point that the global planner only finds paths around large scale obstacles such as rivers and canyons, completely ignoring small scale obstacles. Avoiding obstacles sensed only a short distance from the vehicle will likely require a faster response than a global path planner can provide, so a local navigator or obstacle avoidance behaviour is likely a requirement as well. The local navigator, typically having a faster reaction time than the global planner, can more effectively deal with the smaller scale obstacles and allow the global planner more time to make updates to the global path. The differences between a local navigator and a global path planner are highlighted in Figure 2.

To make a complete navigation system, the UGV also requires a path tracking algorithm to follow the path given by a user or the global path planner. Finally, a decision making module ties all the other navigation components together and decides on the best course of action given competing priorities.



Global Path Planning	Local Navigation
Uses accumulated and a priori information	Uses immediate sensor data only
Concerned with hills, rivers, canyons, forests, roads, buildings, etc.	Concerned with rocks, holes, slopes, bumps, logs, etc.
Plans for long distances and time periods	Plans for immediate vicinity for a short time ahead
Slow, deliberative process	Fast and reactive
Allows robot to avoid getting trapped	Allows robot to travel safely
Plan to reach goal in most efficient manner	Plan to travel as fast as possible
Simple model of vehicle (point robot)	Complex vehicle model (dynamics and kinematics)

Figure 2: *A comparison between the types of UGV navigation.*

2.1 Path Tracking

A path tracker calculates the vehicle speed and steering commands necessary to follow a predefined path accurately, and can profitably be employed in a variety of UGV applications. In a patrol mission scenario, the UGV may be tasked to follow a series of high level waypoints, typically spaced tens to hundreds of meters apart. In another scenario, the robot may be tasked to follow a lead vehicle's trajectory very accurately. A path tracking algorithm should be flexible enough to function in both of these scenarios, working in concert with the other algorithms being used.

A path tracking algorithm should also be robust to changing operating conditions. The errors in the estimates of vehicle heading, position, and speed may vary widely over time, potentially discontinuously. An operator may assign only a single high level goal, or a very detailed track for the vehicle to follow. The same algorithm may be used to control vehicles comprising a variety of sizes and configurations. A good path tracking algorithm should perform adequately under all of these conditions.

Path tracking uses positional information, typically obtained from GPS, IMU, and vehicle odometry to control vehicle speed and steering. In principle, a path could be defined as a continuous function. In practice, however, it is discretized, and described either as a series of straight line segments between interim waypoints or as a series of closely spaced nodes, as shown in Figure 3.

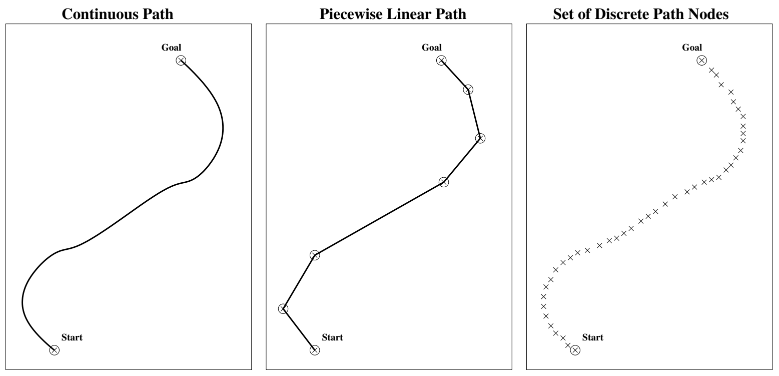


Figure 3: *Various ways of defining the path.*

There are many ways to track paths. If a path is given as a set of waypoints then the simplest is proportional control on the error between the current heading and the heading to the next waypoint, as shown in Figure 4. This method provides goal directedness for many basic obstacle avoidance algorithms [4]. Because this method provides discontinuous control when switching from the current waypoint to the next waypoint as the interim goal, a Proportional-Integral-Derivative (PID) control loop is often used to correct the error to the goal heading.

Even in the absence of obstacle avoidance manoeuvres, the basic heading-to-goal controller as described above is incapable of tracking with arbitrary accuracy a path specified by a series of waypoints. In the event that the vehicle is directed away from the path to avoid an obstacle in its way, having passed the obstacle it will turn back towards the path heading straight towards the interim goal directly from its current location. No attempt is made to track the path itself. A simple expedient which improves path tracking is to steer towards a point on the path a fixed lookahead distance from the vehicle rather than steering towards the next waypoint or the goal position itself. The lookahead point, shown in Figure 4, slides along the path a fixed distance ahead of the vehicle. A PID loop can then be used to control the error between the current heading and the heading to the tracking "carrot", adhering more closely to the intended path. In general, PID control on the heading error is not ideal because it doesn't take into account the steering geometry of the vehicle. Additionally, it can be time consuming to tune the controller gains to generate the desired behaviour.

Using the concept of a lookahead point, a number of other controllers have been implemented, such as Pure Pursuit [5], the algorithm used by DRDC, and described in Section 5. The Control Theory approach [6] executes proportional control on the heading as well as parallel displacement from the lookahead point. The Quintic Polynomial approach [6] fits a polynomial that originates at the current vehicle position and heading, and ends at the lookahead point heading and position. Unfortunately, the system was too complex to be practical, and performance was found to be overly dependant on lookahead distance. The Quadratic Curve method [7] is similar to the Quintic Polynomial except that it makes use of a quadratic curve instead. The Zhang tracker [8, 9], intended for differentially steered robots, has proven effective, but causes oscillations on curved paths. Finally, the Vector Pursuit method [10], which has its basis in screw theory, expands the Pure Pursuit method to operate not only on the displacement to the lookahead point, but also on the desired heading. It seems to perform as well or better than Pure Pursuit, with the expense of added complexity.

Among all these methods, the Pure Pursuit algorithm was chosen for its effectiveness and simplicity. Given that the algorithm would be required to operate under a number of different scenarios, it was desirable that it be robust and simple enough to require little implementation effort. From the experiences at DRDC, the algorithm has proven to be very useful. The algorithm as implemented by DRDC is described in more detail in Section 5.1.

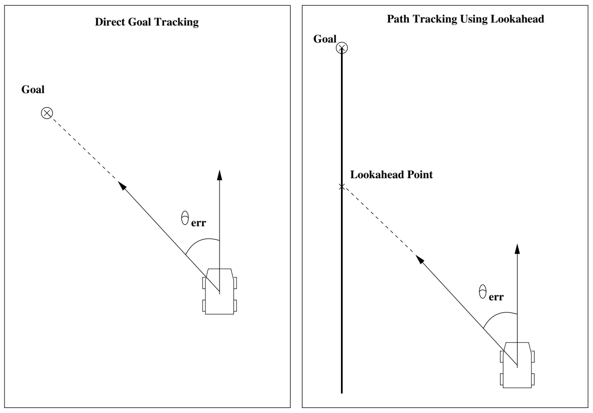


Figure 4: *Direct and indirect PID control for goal seeking.*

2.2 Local Navigation

Local navigation utilizes sensor data to scan the UGV's immediate environment for hazards to avoid while simultaneously seeking a goal location. The term "local navigation" covers a wide variety of topics, including dealing with static and moving obstacles, handling incomplete knowledge of the world, and managing vehicle stability and safety with regards to potential hazards in the environment. Additionally, a local navigation system will have sensors with only limited range to sense the world with. Local navigators generally do not retain acquired information, instead reacting to changing conditions. The maximum safe speed attainable by a UGV is directly related to the speed of operation of the local navigator.

At higher speeds, a UGV will lack sufficient time to plan given limited sensing ranges and rates. One approach to local navigation, detailed physics based analysis and planning, is computationally time consuming, and creates a lower limit for the amount of time a robot needs to react to the terrain, and therefore will limit the speed of a robot reliant solely upon this type of navigation. Conversely, if a system operates very quickly, but without enough detail and fidelity to the real world physics of the terrain and vehicle interactions, it may act inappropriately, hitting objects or becoming unstable.

There are two basic classes of obstacle avoidance techniques. The first comprises methods used by indoor robots in a flat, two dimensional environment to avoid discrete obstacles. The second, physics based modelling and simulation, allows a controller to estimate safe vehicle motions over rough and rolling terrain.

Indoor obstacle avoidance algorithms typically classify the area seen by a robot's sensors in a binary fashion, as obstacle or not, as shown in Figure 5. These algorithms make motion decisions based on the positions of these obstacles relative to the robot [4, 11, 12]. They are simple, iterate quickly and issue new commands at a very high frequency. These methods have also been extended to 3D terrain maps for outdoor navigation [13, 14, 15].

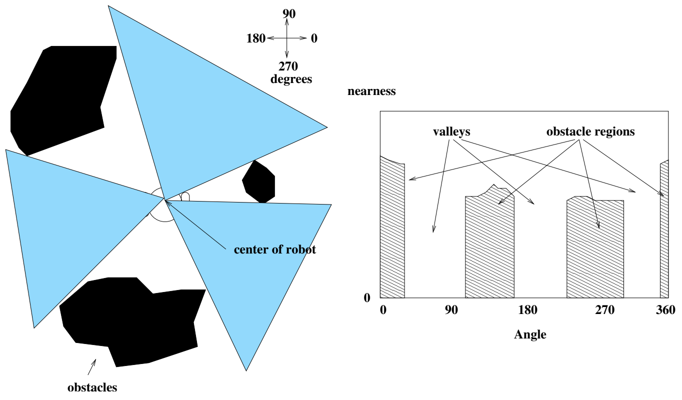


Figure 5: *An example of an indoor obstacle avoidance map.*

The second class of obstacle avoidance algorithms are more complex, using an estimate of the kinematic and dynamic state of the robot platform to plan the vehicle's trajectory around or over obstacles [16, 17]. They are more common for slower moving robots, such as Mars rovers, where execution time is not as critical.

DRDC has adapted two obstacle avoidance algorithms which are based on simple, reactive indoor techniques for use on the Raptor UGV. These techniques are described in detail in Section 5. For a more in depth review of this topic, please see [2].

2.3 Global Path Planning

A human driver looking at a map of a city, countryside or wilderness can quickly and efficiently decide on the best path to a destination. Humans can seemingly autonomically separate portions of the map symbolically, recognize vehicle hazards, identify roads that are more direct, and effortlessly pick the most efficient route. For robots this is not such an easy task owing to their lack of ability to reason symbolically. A robot must first divide up the world into pieces it can recognize as obstacles, undesirable terrain, or dead ends. Then, it must systematically search through the world to find the best route.

Global path planning is the process of deliberatively deciding the best way to move the robot from a start location to a goal location. In more technical terms it is defined as “determining a path in configuration space between the initial configuration of the robot and a final configuration such that the robot does not collide with obstacles and the planned motion is consistent with the kinematic constraints of the vehicle” [18]. The field of path planning borrows heavily from experience in other fields, such as computer networking, artificial intelligence, computer graphics, and decision making psychology. Path planning for UGVs is often based specifically on heuristic search algorithms.

One of the major issues on which path planning research has focused is planning with vehicle constraints. Ackerman steered vehicles are commonly used in outdoor environments. The simple fact that they can't move straight sideways greatly complicates the planning problem.

Typically, vehicle constraints for global path planning are much more important for indoor vehicles than outdoor vehicles due to the differences in scale of the obstacles and the scale of planned paths between the two environments. For outdoor vehicles the boundary obstacles and cul-de-sacs, etc., tend to be of very large scale with respect to vehicle size. Therefore, for planning purposes, planning is done with the robot treated as a point in space. A simplified grid map is shown in Figure 6. A path planner for outdoor UGVs is typically slower, and looks further ahead than those planners for indoor robotics. However, for the purpose of this document, and as a general rule for outdoor UGVs, a global path planner simply plans at a high enough level that the turning radius of the vehicle is small enough to allow the vehicle to negotiate its way around any obstacles it might encounter. Because most outdoor environments in which UGVs operate will be relatively uncluttered this is often a valid assumption. We can rely on the fact that the local navigator will handle the low level task of moving the robot or that the operator will intervene if the robot does get into trouble. However, it must be noted that for many path planning algorithms there is no guarantee that the path planner is competent enough for the environment, and the vehicle may require operator intervention.

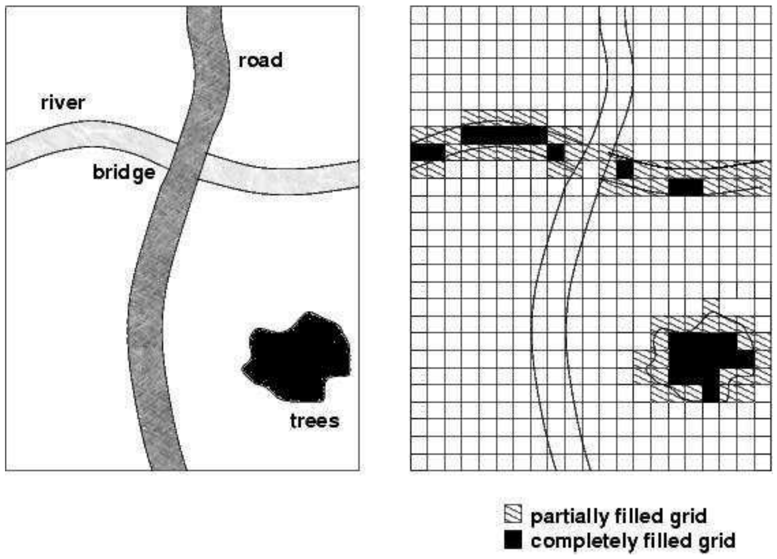


Figure 6: *A simplified version of the grid structure used for global path planning.*

Many search algorithms exist, but the class most suited for outdoor navigation, and selected by DRDC, are based upon the D* algorithms [19]. The DRDC implementation of this is described in Section 5. For a more complete review of this topic, see [3, 20, 21, 22].

2.4 Decision Making

For a UGV to navigate successfully, a number of conflicting priorities must be accommodated. These may be viewed as merely philosophical design directives, but often reflect real criteria for creating reliable, intelligent action. Among these philosophies are debates between deliberative and reactive reasoning, between the use of a compiled world representation or directly using sensor data, and between centralized and distributed control. In general, the debates about these styles of design for intelligent system can be separated into two camps: the behaviour based architectures (reactive, low world representation, distributed control, *ad hoc* construction), and the hierarchical systems (deliberative, complex world representation, centralized control, organized construction). The consensus arrived at among the robotics community is that a successful robot system will have components embodying both sides of these arguments. These are called hybrid controllers, and exemplify the kind of design philosophy DRDC has taken.

2.4.1 Dichotomies in Robotic Navigation

2.4.1.1 Deliberative vs. Reactive Reasoning

Dating back to at least the publication of Rodney Brooks' seminal paper on the subsumption architecture [23], a major debate has existed in the robotics community as to the best architecture for autonomous vehicle navigation. Two schools of thought predominate: deliberative and reactive control. Proponents of the deliberative school design systems that use careful planning, optimal solutions, provide for higher level goals, avoid mistakes, and are very goal oriented. The weakness in this ideology is that deliberative reasoning often takes too long. If the result of a step is not what was expected, or the world changes in the meantime, the entire plan must be regenerated. As a response to these problems, proponents of the reactive school counter with systems that attempt to map input more directly to output, bypassing the planning stage. They take a "feedback" approach, rather than "feedforward". These systems are more robust to changing and unknown environments, and are easier to design, but are often sub-optimal, and are more difficult to make goal oriented.

2.4.1.2 World Representation vs. Sensed Conditions

Closely tied to the previous dichotomy is the debate between symbolic internal representation, and direct reaction to the environment. The reactive school tends to avoid the representation and objectification necessary to create a complex internal model of the world. According to this view, the creation of world models is a computationally heavy bottleneck that can be avoided: "let the world be its own model." This method can break down when sensors are unreliable. It also makes it difficult for the autonomous agent to remember hazards or alternate avenues that it has encountered. On the other hand, fusing information and providing a symbolic representation of the world has a number of important benefits: it allows more complex "look ahead" decision making, such as path planning, and allows the elimination of noise and the combination of evidence. There is also the opportunity to create world models with state, eg. the trajectory of a moving obstacle can be estimated including its position at some point in the future. There is inherent overhead and complexity introduced, but this may be offset by the potential synergy created between the sources of information. However, if the world model should get out of synchronization with the real world with respect to time, position or accuracy, a UGV solely reliant on the world model could run into an obstacle in an area it believes to be free. As well, it is difficult to create a system which has more symbolism and representation, yet remains flexible. For these reasons, as for the previous dichotomy, the consensus is that balance and complementarity are required between these ideals in order to create intelligent systems. If the world model should fail, then we need to fall back to a more direct mapping of sensing to action.

2.4.1.3 Information Sharing Between Sub-Modules

The reactive school also advocates a lack of communication between modules. Each of the modules has its own window on the outside world, and acts according to its own information. In the end, there is a method for arbitration between the intelligent sub-modules, but there is no passing of information between the control processes. There are great savings in complexity and time to be gained as compared with designs which share a lot of information. This debate is very closely tied to the previous one regarding internal models: none of the benefits expressed above with regards to world representation are possible without internal communication, but the added complexity can be cumbersome.

2.4.1.4 Centralized vs. Distributed Processing

Processing for a mobile robot can be distributed with respect to processor threads of execution, separate processes on a single processor, on many separate processors, or even physical location. Traditional centralized systems tend to be coherent and easier to understand, but are subject to catastrophic failure. On the other hand, distributed systems tend to be more robust and flexible, and can take advantage of the processing power of multiple computers. However, they tend to be harder to constrain and more difficult to understand.

2.4.1.5 Monolithic vs. Arbitrated Control

Traditional control theory focuses on a single, monolithic controller which is the only source of commands to the system under control. This approach has been successful at many tasks, exhaustively demonstrated on non-mobile robots such as industrial processes, machines, and robotic arms. Input is continuously monitored by one controlling process, and the control signal used is the value decided upon by this controller. However, this approach provides major difficulties when we attempt to scale up to the complexity of mobile robots. Intelligent behaviour in a mobile robot, especially in outdoor environments is not simply a matter of designing the perfect control system. The environments they operate in are too complex to enable us to design this system. A more recent trend in mobile robotics is to provide control as a discrete set of behaviours which are themselves monolithic, independant control loops. It has been demonstrated that these behaviours may be sequenced, switched or arbitrated to create emergent intelligent behaviour, even if the sub-modules on which they are based are not very intelligent.

2.4.1.6 Combined Action vs. Single Chosen Action

Having decided to combine multiple simpler behaviours or controls to form the intelligence in a vehicle, how does one accomplish this? This quandary strikes to the heart of artificial intelligence and decision making research. Many methods have been proposed, some choose a summation of actions to define a resultant behaviour, while others believe that switching between behaviours, selecting the most appropriate for the current situation, is more advantageous. For example, Rodney Brooks's subsumption architecture [24, 23] simply hard codes a priority scheme into the system based on sensor inputs, resulting in a series of single chosen actions. As an extension of this, many have suggested creating intelligence by having an intelligent module choose a set of sequenced single actions which create a complex behaviour [25]. In contrast to this philosophy, the motor schemas [26] and potential fields [27] architectures sum up the priorities of the vehicle to create a resultant priority. It is assumed that when a priority becomes strong enough it will completely take over control of the vehicle smoothly, rather than in an abrupt switched manner. Modern extensions of these two dichotomies have attempted to have it both ways. The DAMN architecture [28] uses a complex method of arbitration in which two of the control agents arriving at the same decision, will have their outputs blended, but if there is opposition, one agent's output will take priority over the other.

2.4.2 Modern UGV Controllers

2.4.2.1 Hierarchical Systems

Hierarchical systems, which include most of the earliest forms of robot artificial intelligence (prior to 1985) [29], have a number of distinct characteristics. They usually follow the sense-model-plan-act paradigm. They use sensor data to model the world very explicitly (often in the form of a map), plan within this map space, and pass the planner's instructions to a lower level controller responsible for carrying out the action. This type of controller uses all of the available knowledge of the world, past and present, but may be too slow to react to changes in the environment. These controllers are useful because they provide structure and ordering of the relationships between sensing, planning and acting, but are unwieldy, considering that the world model needs to be refreshed on every update cycle, and that frequent replanning episodes are required in the face of changing conditions. Sensing and acting are always separated and quick reaction is not possible. These are especially important concerns for UGVs operating at higher speeds. Basic examples of hierarchical systems can be found in [30, 31, 32]. The NIST architecture for UGV control, the Real-Time Control System (RCS) [33, 34, 35] has also adopted this design philosophy.

2.4.2.2 Behaviour Based Systems

As a response to stagnation in the progress of this type of control, a group of roboticists began to shift emphasis from planning to sensing, called "behaviour based control"[36], in which a number of agents within the software architecture compete for control of the vehicle. There is often very little combined world representation, and each behaviour typically reacts to a single sensor or acts toward a single goal. These systems are very quick to react to changes in the environment, but lack the foresight necessary to complete complex tasks. The word behaviour comes from study of biological behaviours in which a set of sensory inputs directly translates into a set of actions. Behaviours can be concurrent, executing at the same time and reacting to a variety of stimuli. Unfortunately, the concept provides no indication as to an appropriate manner in which to combine these actions to execute concurrent tasks. Much of the work in this field has focused on mechanisms for combining these behaviours. The outcome of combination can vary markedly: the actions may balance each other out, sum together, or one may possibly dominate the other. Controlling the effects of this problem is one of the central issues with the reactive, behaviourist school, and reflects very closely the "one chosen action" vs. "combined action" dichotomy presented in the previous section.

Behaviour based design philosophies have a number of benefits. They follow good design principles of modularity, minimizing coupling and maximizing cohesion, allowing for easy development and testing of individual behaviours. Behaviours can be developed in an *ad hoc* fashion, and new features are easily added. These systems are also much better at reacting quickly to changing environments, as the link between sensing and acting is tighter. The major detraction to these systems occurs as complexity increases: interactions between behaviours increase until it becomes difficult to predict the system's behaviour. Examples of this type of system can be found in [23, 24, 25, 37]. A good overview can be found in [36].

2.4.2.3 Hybrid Systems

In early 90's, it became clear that planning has its place in a robotic system, and that discarding it entirely was too rash. Researchers attempted to put deliberation and planning back into mobile robots, while keeping the advances made by the reactive behaviour based school. In general this requires asynchronous processing: a slower deliberative planner operating concurrently with a faster reactive system, which is much easier with modern computer operating systems with multi-tasking and multi-threading. The result is a more distributed design style, with a planner separated from real-time control. Sometimes higher logic also provides monitoring of the lower level behaviour's progress (Are the wheels spinning? Are we stuck in a corner?), a deliberate attempt at providing system robustness.

In hybrid control, sensors contribute to the individual behaviours, but also to a global world model used for planning. The deliberative portion works with symbolic knowledge, while the reactive portion works more directly with sensor data. Sensor data is routed to behaviours, but also to the planner to accumulate and store. Some current examples of hybrid robot controllers can be found in [38, 39, 40].

An important contribution to UGV decision making is the DAMN Architecture, developed by Rosenblatt [28]. DAMN, a Distributed Architecture for Mobile Navigation, uses a voting scheme in which any number of modules, including both deliberative and reactive components, can contribute to a centralized module which decides on an output behaviour. All of the voters can operate asynchronously, and the system only considers actions which are immediately realizable. It takes into account previous decisions (to provide hysteresis and persistence), but re-decides several times a second. It makes no constraints on deliberative/reactive nature of the modules, and the process that arrives at each vote can be arbitrarily simple/complex. The behaviours are decentralized, which avoids systems bottlenecks, and increases flexibility and robustness, but the decision making component is centralized, providing coherence and tractability.

The centralized decision maker, or arbiter, uses many different modes:

- Constraint Mode - For a variable such as UGV speed, all the behaviours submit their lowest acceptable speed. The value chosen is the lowest value acceptable to all the behaviours.
- Actuation Mode - The arbiter chooses between many possible options, all of which are voted on by the behaviours. An example of this may be turning angle where the behaviours vote on a set of candidate arcs for the vehicle to travel.
- Effect Mode - The arbiter chooses between a variety of output effects, chosen from an abstract command space, such as field of view for camera.
- Utility Mode - Each behaviour votes on the outcome it wants, such as the best future vehicle location, and the arbiter decides on a control strategy which accomplishes the most popular outcome.

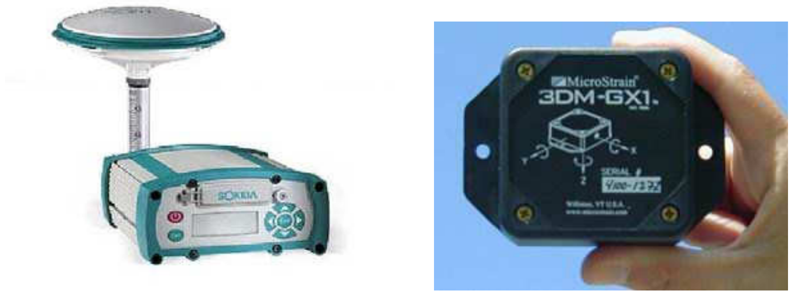
The centralized decision maker uses a mode manager to apply weights to each behaviour to make a decision, almost like a subsumption system where one takes precedence over the other, but which are adjustable. In this way, it can easily add or drop a behaviour, while simultaneously completing complementary goals. Extensions to the basic DAMN architecture to account for uncertainty in voting, based on utility theory, have also been suggested. In addition to the voted action, each vote contains a certainty of decision, as well as the utility of the decision. The DAMN Architecture has proven to be effective on many practical UGVs [41, 42] and was used as the basis for the decision making architecture developed by DRDC, described in Section 5.6.

Further summaries in the area of robot decision making can be found in [20, 43]. There are also many other methods not described here, such as fuzzy logic [44], or Markov Decision Processes [45].

3 Vehicle Platform

DRDC selected the Koyker Raptor as its UGV demonstration platform, Figure 1, based on payload and drivetrain requirements. In each of two vehicles at DRDC, a 25 hp gasoline engine powers a 4x4 hydrostatic drivetrain while generating an additional 1.5 kW of on-board power. The vehicle's on-board intelligence enclosure houses power inverters, quad and dual Pentium servers, ethernet and USB hubs. MeshDynamics radios provide 802.11b class wireless communications. XJ Design of Ottawa, Canada converted the vehicles to drive-by-wire using an on-board MPC555 microcontroller.

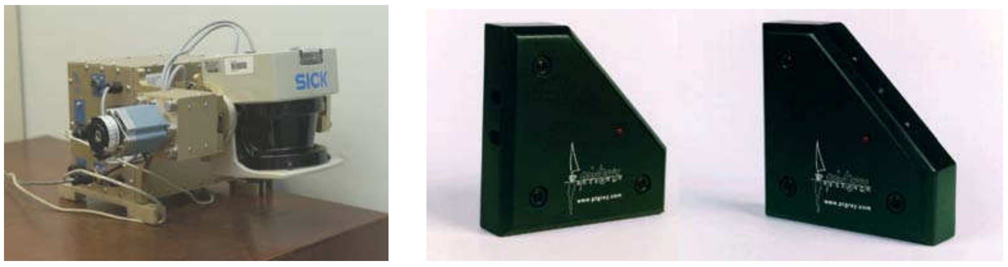
The sensing systems on the Raptors consist mostly of commercial, off-the-shelf hardware, including a GPS, an IMU, and SICK laser range finders (LRFs). The Sokkia GSR2600 GPS, shown in Figure 7(a), combined with a Pacific Crest PDL radio, supplies differentially corrected GPS positions with an accuracy of $2-5$ cm at an update rate of 4 Hz. The IMU employed is a Microstrain 3DM-GX1, see Figure 7(b), which uses magnetometers, gyros, and accelerometers to produce orientation and angular rates with respect to gravity and magnetic north. Because of the magnetic effects of the vehicle chassis, it was necessary to perform the Microstrain hard-iron calibration procedure to get reliable orientation data. Odometry data is collected by the MPC555 vehicle controller, using Hall-Effect sensors at the wheels.



(a) (b)

Figure 7: *The Sokkia GSR2600 and Microstrain 3DM-GX1 sensors.*

To overcome the bulk and expense of traditional 3-D LRFs [46, 47], DRDC turned to inexpensive, light weight 2-D LRFs such as the SICK laser range finder. DRDC and Scientific Instrumentation Ltd. developed a nodding mechanism for a 2-D SICK LMS211, creating a system that returns 3-D data, shown in Figure 8(a). The laser measures the time of flight for light pulses in a 180 degree horizontal swath and internally converts the time value into the corresponding distances. Communicating through an ethernet interface, the embedded RTEMS controller nods the laser from $2-90$ degrees/s with 0.072 degree resolution and 4 cm accuracy over $1-30$ m [48].



(a) (b)

Figure 8: *The Nodding Laser and Digiclops sensors.*

For stereo vision, DRDC adopted Point Grey's Digiclops system to provide high speed range image streams. The Digiclops develops a disparity map between three camera image streams, publishing the resulting 3D range image stream over an IEEE-1394 Firewire digital connection. The Digiclops is capable of producing disparity maps with a resolution of

640×480, but on the Raptor UGV the Digiclops' performance is downgraded to a resolution of 320×240. Figure 8(b) shows the Digiclops stereo system.

4 DRDC Architecture

4.1 Software Design Principles

The DRDC Navigation and Decision Making Architecture is a hybrid controller which mixes traditional model based, deliberative methods (mapping, world representations, path planning), with reactive elements (sensor based obstacle avoidance), combined by an arbiter.

Figure 9 shows a high level overview of the navigation and decision making software used on the Raptor UGVs. The specific modules which are the topic of this report are the *Planning and Goal Seeking* modules, and the *Decision Making* module. As can be seen, there are also many peripheral software modules required to complete the UGV system.

Each of the blocks shown consists of an independant computer program, or process, communicating via CORBA (over TCP/IP) with the other software modules, which can be spread across a number of computers if more processing power is required. Data is generally made available in a publish/subscribe fashion, and data flows are indicated with the arrowed lines. When a process starts, it connects to a central Naming Service, that informs any process of the data items available and the location of the desired modules. Thus, each module can run at any location on a network, and each data flow item publishes to any subscriber that requests its data. The data flows are almost exclusively "Event Driven", meaning that they are published to the requesting process as soon as the data is ready. In this way, the system can be as reactive as possible to new information.

Each of the data flow events conforms to a pre-defined IDL (Interface Description Language) format. For example, data events of the type "Gps", will conform to the same format, regardless of the hardware sensor used to produce the event.

As a concrete illustration of the communication process, consider the flow of laser range data from the sensor to each of the behaviours. The block marked "Laser" in Figure 9 handles TCP/IP communication with the SICK laser scanner on the Raptor UGV. When it is started, it initializes the sensor, and registers itself with a Naming Service as "Laser", a pre-defined IDL interface type. When other modules are started, such as the *Terrain Map*, or *Obstacle Detection*, they consult the Naming Service to find the interface "Laser" which produces "Range3dLaserEvent" data, and then subscribe to those data events. When the laser range sensor begins producing data, it is delivered to those subscribed processes as the data becomes available, to be used as those processes see fit.

This control architecture is extremely modular. This has a number of very practical advantages:

- Separation of hardware and software components - Any of the hardware sensors can be swapped for another sensor, without any change to components subscribing to that data type. For example, many different types of GPS receivers could be used to produce “GpsIDL data”, and therefore the many types of receivers are interchangeable from the point of view of a subscribing process.

- Contribution by many researchers - Scientists can easily collaborate because of the pre-defined interfaces and minimal amount of coupling between the modules.
- Testing and debugging - Each piece of the architecture can be evaluated on its own through the use of pre-recorded input events and confirming the data events generated.
- Experimentation - Any algorithm can easily be substituted for another for comparison purposes.
- Configurable behaviour - A quick change of vehicle behaviour is possible simply by enabling/disabling certain components. For example, if the system was intended to simply safeguard tele-operated control, only the *Obstacle Detection* module would need to be activated. As more autonomy is desired, additional behaviours can be enabled.

Additionally, the asynchronous publish/subscribe data delivery method has a number of benefits for practical UGV systems:

- Reactivity - The system can respond to information as soon as it is available, rather than waiting for the next processing iteration.
- Distributability - Processes can be moved to where computing power is available.
- Flexibility - It is very easy to substitute different sensors on the vehicle, or migrate the system to a different platform.

In addition to the event driven publish/subscribe model that is used to communicate most of the data in the system, most modules also have a “Polled Interface” by which other modules can make a function call (over the network) to retrieve the latest data at their convenience. More detailed information on this architecture can be found in [1, 49].

4.2 Middleware

In order to facilitate this modularity and network flexibility, a software toolkit called “Miro”[50] is used. This middleware handles the delivery and receipt of data events between modules over the network, using the pre-defined IDL interfaces and CORBA protocols. Miro was chosen after a thorough review of available systems [51], and is flexible enough to provide for the needs of the autonomous robot and multi-robot systems at Defence R&D Canada – Suffield well into the future.

In addition to event driven publish/subscribe data delivery, Miro also offers a number of other useful tools, such as features for RS-232 and TCP/IP communications, multi-threaded processing, timers, etc. It also includes a “Log Player” which allows the recording

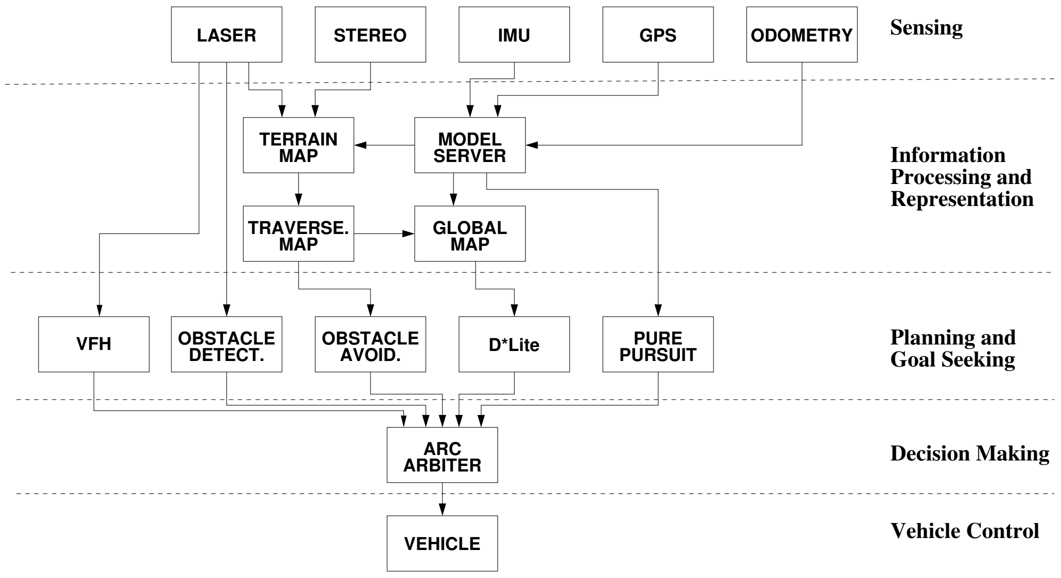


Figure 9: *The complete Raptor software flow diagram. Each box represents a service that is its own process in the operating system, and can reside anywhere on the network. Each line represents a pre-defined interface, with data flows accessible to any subscriber.*

and playback of any of the data events. This is invaluable for debugging and testing of individual modules. More detailed discussion of the software middleware which binds this architecture together can be found in [52, 53, 1, 54].

4.3 Software Modules

Each of the components shown in Figure 9 will now be described to give the reader an idea of how the overall system works. The data format (IDL interfaces) used for the modules described can be found in Appendix A.

4.3.1 Sensing

As described in Section 3, the Raptor platform has a wide variety of sensors which can be used to produce autonomous behaviour. Each of these sensors has its own "driver" module which will communicate with the hardware (TCP/IP, RS-232, etc.) and will package the data into the pre-defined IDL format for that type of sensor. This may include some data processing, such as transforming laser range readings into 3D positions relative to the scanner.

4.3.1.1 Laser and Stereo

The *Laser* component uses TCP/IP to interface to the the nodding SICK laser scanner, and generates 3D terrain points based on the range readings and the current scanner nodding angle. This data is only accurate with respect to the scanner itself, and will later be corrected for the vehicle pose. Once a 180 degree scan has been retrieved from the scanner (every 26.6ms), the Laser component publishes a “Range3dLaserEvent” to each of its subscribers. The *Stereo* component is very similar to the *Laser* component, except that it interfaces to the the Digiclops stereo vision camera, and has a slightly different data output format.

4.3.1.2 IMU

The current version of the *Imu* component retrieves roll, pitch and yaw data via RS-232 from a 3DM-GX1 inertial measurement unit, and publishes an “ImuIDL” event every 100ms.

4.3.1.3 GPS

The *Gps* component uses RS-232 to interface with the Sokkia GSR2600 GPS, which in turn receives differentially corrected signals from a Pacific Crest PDL radio. This information is updated at 4Hz, at which time the *Gps* component produces events following the “GpsIDL” interface.

4.3.1.4 Odometry

This component reads odometry data produced by the MPC555 controller and the wheel encoders on the Raptor UGV, and produces “PositionIDL” events at a rate of 4Hz. This component is actually part of the *Vehicle* module, but can logically be thought of as a sensor input.

4.4 Information Processing and Representation

The components described in this section take sensor data produced by the previously discussed components and process them into a world representation useable by the various navigation and decision making components on the Raptor UGV.

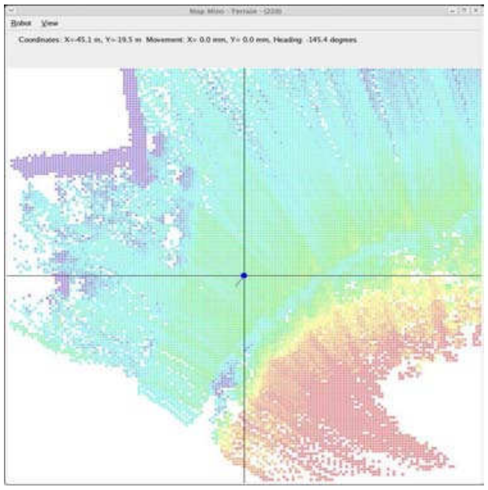
4.4.0.5 Model Server

Navigation of large UGVs requires a variety of sensors which are widely distributed across the vehicle, and subject to the pitch, roll, and yaw of the platform. The *Model Server* component maintains the vehicles position and pose, and then establishes the position of each of the sensor components, allowing the other software modules to adjust the data returned from the sensors. The *Model Server* consumes data from the *Gps*, *IMU*, and *Odometry* modules to create this pose estimation, creating a “PoseTransformIDL” event, typically 10Hz. This component is described more fully in [55].

4.4.0.6 Terrain Map

In order to navigate unknown terrain effectively, UGVs must be able to create an accurate representation of the operational environment. Typically this is done by constructing a geometric representation of the environment, called a *Terrain Map*, from exteroceptive and proprioceptive data streams. Similar types of maps can be found in [56, 57, 58, 59, 60].

The DRDC *Terrain Map*, shown in Figure 10 uses 3D range data from the laser and stereo sensors' "Range3d" events to create a $2\frac{1}{2}$-D representation of the world. It also uses the "PoseTransformIDL" events to correct the range sensor data for roll, pitch, yaw and position of the platform when the sensor data was acquired. The service fuses range data into a grid map, a rectangular array of regions, to build a $2\frac{1}{2}$-D or digital elevation map [61, 58, 59]. The *Terrain Map* estimates the height of the world at each cell in the 20cm x 20cm grid, as well as the elevation variance. The data which goes into the terrain map is kept for a period of time, but is statistically combined so that old, or less certain data will be less important to the estimate of terrain height in any given cell.



Coordinates: X=-45.1 m, Y=-19.5 m Movement: X= 0.0 mm, Y= 0.0 mm, Heading -145.4 degrees

Figure 10: *Typical Terrain Map. Red cells are lower elevation (a ditch), blue higher (the wall of a building), and green indicates flat, passable terrain. The direction of vehicle travel is indicated by the small line on the dot in the center of the image.*

This service maintains two types of map. The standard *Terrain Map*, as shown in Figure 10, is centered on the vehicle and is referenced to the cardinal directions (N,E,S,W), and North points towards the top of the map. The *Egocentric Terrain Map*, shown in Figure 11 is local to the area directly in front of the vehicle, scrolling and wrapping as the vehicle moves in both the X and Y directions. Whenever new range data is available and has been fused into the Terrain Map a "MapArrayEventIDL" and an "EgoMapArrayEventIDL" event are published. This typically is set to produce new events every 500msec.

More information on this component can be found in [62].

4.4.0.7 Traverse Map

Maps used for robot navigation are also capable of representing obstacles and other features, such as hills, slopes, bumps and dips, found in outdoor environments. The *Terrain*

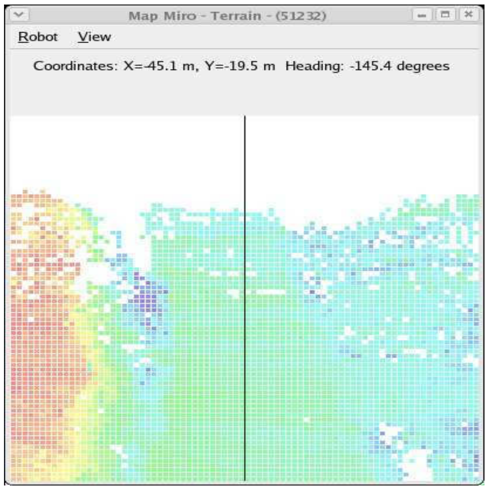

Map Miro - Terrain - (51232)
Robot
View
Coordinates: X=-45.1 m, Y=-19.5 m Heading: -145.4 degrees

Figure 11: *Typical Egocentric Local Map*

Map, which encodes the geometry of the environment, can further be analyzed to provide a measure of the traversability of the terrain. This reduces data dimensionality and provides more useful data to path planners. For the purposes of this paper, these reduced dimensionality maps are referred to as *Traversability Maps*. The resulting *Traversability Map* provides a useful metric for path planning and obstacle avoidance algorithms to determine the "best" path to follow.

The *Traversability Map*, depicted in Figure 12, contains a *Goodness* rating of each grid cell based on the terrain hazards in a given area, similar to the Gestalt system [57]. For this map, the grid cells are typically enlarged to 50cm x 50cm. It interprets geometric data by calculating statistics within the *Terrain Map*, such as roughness, pitch, step height, etc. These statistics can then be compared to vehicle specific thresholds to determine if the grid-cell is traversable, impassable, or unknown. In doing so, the Traversability Map interprets geometry from a vehicle specific context (i.e. a particular area may be traversable for a large UGV but impassable for a smaller UGV). This provides portability of the algorithm to different platforms, only requiring that the user change the vehicle specific parameters.

The *Goodness* rating of any cell is given as the following:

- Unknown cells are assigned a goodness value of 2.
- Untraversable (Obstacle) cells are assigned a goodness value of 1.
- Traversable cells are given a goodness rating between 0 and 1 indicating how desireable the position is, based on its roughness, step height, pitch, etc.

In addition, each *Traversability Map* cell also contains a *Confidence* rating between 0 and 1, which indicates the certainty with which the traversablity was calculated.

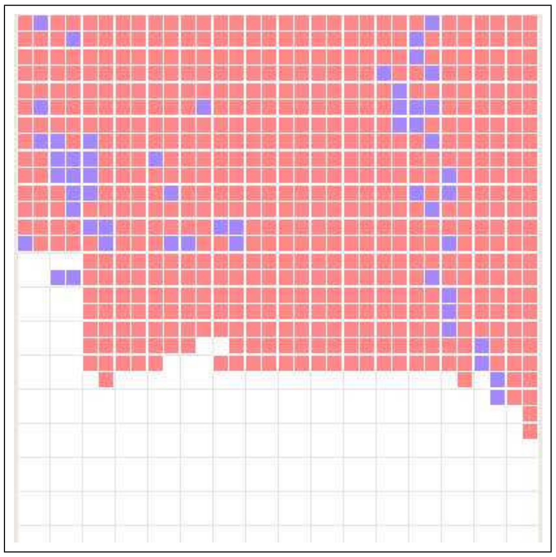


Figure 12: *A Traversability Map. Blue, red, and white cells indicate impassable, traversable, and unknown cells respectively. The area enclosed in the rectangle roughly corresponds to the Ego map.*

The *Traversability Maps* are generated whenever a new *Terrain Map* is generated, typically every 500msec, as a "TravMapArrayEventIDL" event. More information on this component can be found in [63].

4.4.0.8 Global Map

The *Global Map* component maintains a cumulative view of the traversability information generated by the *Traversability Map* component, described in the previous section. It subscribes to "TravMapArrayEventIDL" events component and generates "PlanMapEventIDL" events. The "PlanMapEventIDL" events simply encapsulate the information contained in the "TravMapArrayEventIDL" events received by the *Global Map* component.

In addition, the *Global Map* component employs a polled interface for use by the *FindPath* component in planning a path. This interface enables:

- waypoints for a path to be set and queried in the same fashion as the *PurePursuit* component handles waypoints, described in Section 5.1,
- a planning space boundary to be set and queried, and
- the vehicle's current location and the cost of traversal to adjacent locations in the planning space to be queried.

4.5 Vehicle Control

The *Vehicle* module is primarily a translator, which accepts translational and rotational velocity commands from the Arc Arbiter, and translates them into the RS-232 commands to the vehicle MPC555, in terms of steering angle and velocity. It can be thought of as a "personality module" or a "hardware driver" for each specific platform that the navigation and decision making architecture will work on. In addition, this module is also responsible for collecting information such as vehicle odometry, generating the appropriate data events, and monitoring for unsafe driving conditions (high engine temperature, etc.).

For a vehicle moving in an arc, it can be shown that the arc curvature, $\gamma = 1/radius$, can be defined in terms of its rotational and translational velocities. These values for rotational velocity, ω, and translational velocity, v_{set}, are passed to the vehicle controller, which converts them to a steering angle and drive speed. The relationship is as follows:

$$\begin{aligned} \gamma &= \frac{\omega}{v_{\text{set}}} \\ \omega &= \gamma \times v_{\text{set}} \end{aligned}$$

The average angle of the front wheels, δ, in a vehicle of wheelbase, L, with Ackerman steering (Figure 13) as a function of path radius of curvature, R, is given by Gillespie [64]. Gillespie has shown that

$$\delta = \frac{L}{R}$$

and from above

$$R = \frac{1}{\gamma} = \frac{v_{\text{set}}}{\omega}$$

therefore

$$\delta = \frac{L \times \omega}{v_{\text{set}}}$$

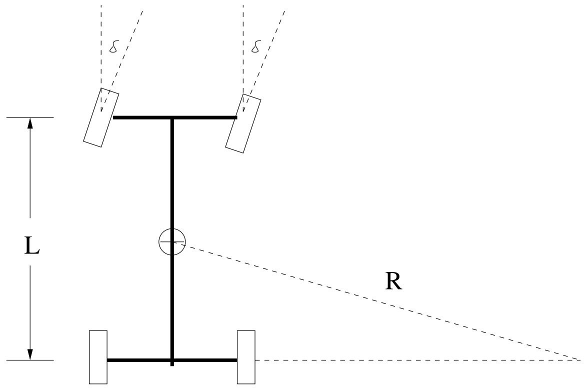


Figure 13: *The Ackerman geometry used to control the Raptor vehicle.*

At this point the steering angle δ and the drive speed v_{set} are passed to the vehicle PID loops for control, via the RS-232 interface.

4.6 User Control Station

Any navigation and decision making system also needs the ability for the user to supply high level goals to the robot. An example scenario would have a mission commander tasking the robotic vehicle with a patrol mission: the path to be followed is described as a series of waypoints tens or hundreds of meters apart, which the autonomous vehicle is to navigate between without operator intervention. In order to accomplish this, DRDC has developed a user control station based upon the Barco ODS Toolbox, which operates on a remote laptop computer, and connects to the vehicle via its 802.11 mesh radio network.

The *Control Station* is shown in Figure 14, in which a set of waypoints (red circles) is shown on the map display of part of the Defence R&D Canada – Suffield range. Waypoints are sent to the robot as a set of positions, in latitude and longitude coordinates, accompanied by a radial tolerance for each. For a waypoint to have been reached, the robot must pass it at a distance less than or equal to the radial tolerance for that waypoint.

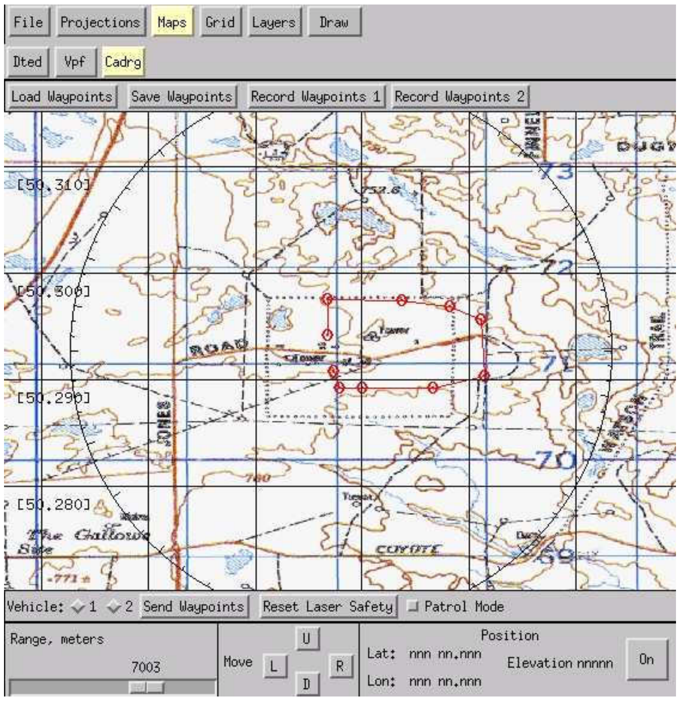


Figure 14: *A high level path supplied via a user control station.*

The sequence of latitude/longitude waypoints are passed from the *Control Station* to the individual algorithms which perform goal-directed behaviour, such as the *D* Lite*, *VFH* and *Pure Pursuit* modules. It uses the *VehiclePlan* interface to do this, described in Section 5.1. The *Control Station* can also pass special operating modes to the vehicle (e.g. to ignore obstacles in its path or to continuously loop through waypoints, etc.) In addition, the *Control Station* subscribes to the “Gps”, “Imu” and/or “Pose” events, in order to update the current position, speed, etc of the vehicle for the user. It is capable of monitoring/controlling multiple UGVs at the same time. The interface by which the waypoints are communicated is given in Appendix A.

5 DRDC Algorithms

Having laid the groundwork of the architecture and the supporting systems, the discussion will now turn to the heart of the navigation and decision making software: the individual algorithms of which it is comprised:

- *Pure Pursuit* - A path tracking algorithm to allow the robot to seek user-defined waypoints.
- *Obstacle Detection* - A simple method to use raw laser range data to safeguard the vehicle during either tele-operated or autonomous control.
- *Vector Field Histogram* (VFH) - This 2D algorithm provides reactive avoidance of discrete stationary and moving obstacles, while seeking goals.
- *Obstacle Avoidance* - This 3D algorithm analyzes terrain to avoid steep or rough areas.
- *D* Lite* - A deliberative path planning algorithm to provide long term goal seeking behaviour.
- *Arc Arbiter* - The decision making component which compiles the output of the other algorithms into an intelligent choice for vehicle speed and steering.

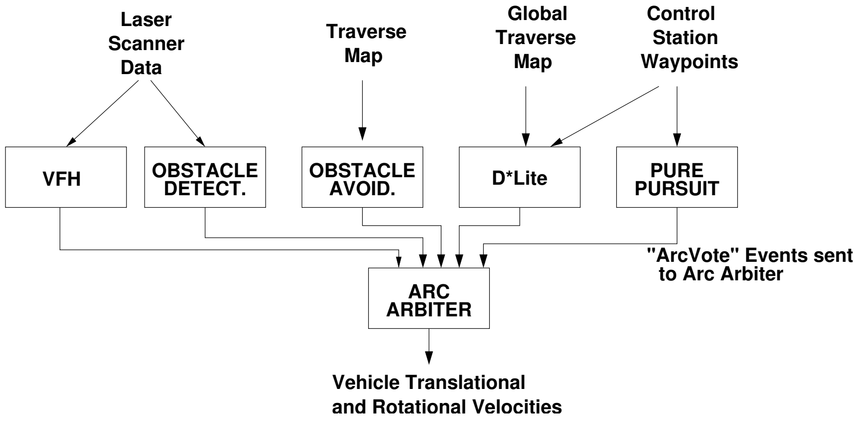


Figure 15: *Simplified software flow diagram of the navigation and decision making components.*

A simplified diagram of the navigation and decision making components is shown in Figure 15. Each of these asynchronous behaviour components operate on different input data to provide a sliding scale of autonomy, as only those components necessary for the task need to be activated. Each independent behaviour publishes CORBA “ArcVote” event objects, consumed by the *Arc Arbiter* to direct the Raptor’s movement. A summary is provided in Table 1. It shows the components name, the data events it subscribes to, the interfaces it polls to get data, the events it publishes, the interface it presents for other modules to poll, as well as its update rate. The following sections describe five behaviour components implemented thus far in detail.

Component	Subscribed Events	Polls	Published Events	Polled Interface	Update Rate
FindPath	-	GlobalMap	ArcVote	VehiclePlan	$\geq$ 500 ms
Obstacle Avoidance	TravMapArrayIDL	-	ArcVote	-	500 ms
Vector Field Histogram	RangeGroupEventIDL, GpsIDL, ImuIDL	-	ArcVote	VehiclePlan	500 ms
Pure Pursuit	GpsIDL, ImuIDL	-	ArcVote	VehiclePlan	100 ms
Obstacle Detection	Range3dLaserIDL	-	ArcVote	LaserSafety	26.6ms

Table 1: *Planning and Goal Seeking Components*

As will be described in Section 5.1, the components which provide goal directed behaviour receive their waypoints via their “VehiclePlan” interface, which is polled from the user control station.

Each of the components in Table 1 interfaces to the *Arc Arbiter* by publishing an “ArcVote” event, based upon the “ArcVoteIDL”, to the event channel. Each “ArcVote” event expresses the behaviour’s desire to travel on each arc from a pre-defined set of candidate arcs (i.e. vehicle steering angle and speeds). Each behaviour can also veto any arc that it chooses, to ensure that the vehicle will not travel that path. Each behaviour also provides a maximum speed that the vehicle should be allowed to travel for each candidate arc. The votes from the various active behaviours will be combined in the decision making process, described in Section 5.6, into a vehicle action. The structure of the *ArcVote* interface for a single arc is shown in Table 2. The “ArcVote” contains an array of these votes, one for each of the candidate arcs, as well as an indication of which behaviour generated this “ArcVote” event.

The typical distribution of these candidate arcs is shown in Figure 26.

5.1 Pure Pursuit

The *Pure Pursuit* component allows the Raptor UGV to seek high level goal GPS way-points provided by the human controller, by attempting to follow the straight line segments between them. It subscribes to the “Pose” event objects published by the *Model Server*, executes the *Pure Pursuit* algorithm and publishes an “ArcVote” event object. The algorithm continually calculates the candidate arc necessary to return the vehicle to the ideal path, at a specified look-ahead distance. This results in smooth path following behaviour, as is illustrated in Figure 16.

Item	Type	Description
Curvature	float	The curvature, in units of $1/meters$ of the candidate arc
Desirability	float	The vote: a number between 0 and 1 indicating the desirability of the arc
Certainty	float	A number between 0 and 1 indicating the behaviour's belief in its data
MaxSpeed	float	The maximum speed at which the behaviour finds it acceptable for the vehicle to travel this arc (m/s).
Veto	bool	Setting to true vetoes this arc

Table 2: *A vote for a single candidate arc*

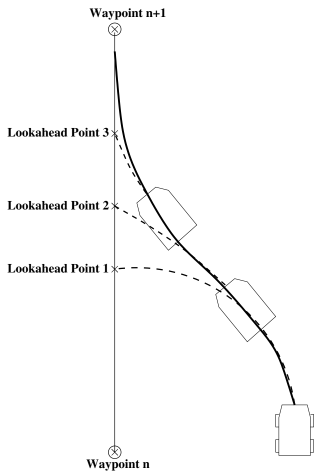


Figure 16: *Illustration of the method Pure Pursuit uses to follow paths.*

The waypoints given to the vehicle from the *Control Station* are passed to the *Pure Pursuit* component via its polled interface defined in the "VehiclePlan" interface, shown in Table 3. The "WaypointGroupIDLs", referred to in the table, consist of arrays of latitude/longitude pairs, which define the sequence of waypoints to follow. The "PatrolMode", defined in the table, indicates whether or not the vehicle should continue to cycle through those waypoints after it has completed the original sequence.

The standard implementation of the *Pure Pursuit* algorithm in the literature uses a detailed path described as a set of closely spaced nodes. However, for many UGV applications, the user will not want to specify an extremely detailed path. In the waypoint implementation of *Pure Pursuit* described here, each of the waypoints passed to it is considered an interim goal location and the basic algorithm tracks the straight line segments between consecutive pairs of waypoints. A list of all the waypoints is maintained, but at any given time the algorithm is only operating between the current and the next waypoints.

Poll Interface	Description
setWaypointList(WaypointGroupIDL)	Assigns waypoints to the vehicle
WaypointGroupIDL getWaypointList()	Retrieves the vehicles current working waypoint list
WaypointIDL getCurrentWaypoint()	Retrieves the waypoint that the vehicle is trying to reach
setPatrolMode(boolean)	Tells the vehicle to loop through the waypoints, or not
boolean getWaypointList()	Reports whether or not the vehicle is in Patrol Mode
setHalt(boolean)	Stops the vehicle if set to true (i.e. veto all arcs)

Table 3: *Polled Interface for Waypoints in Pure Pursuit*

The source of positional information is not necessarily important to the algorithm, however, it does require information about both position and heading. The sensor used for this implementation was the Sokkia GSR 2600 GPS receiver, which outputs heading and positional information from either differential or standard GPS readings at a rate of four times per second.

This implementation of the algorithm can be summarized as follows:

1. The implementation waits until it receives a path in the form of a waypoint list from the operator before moving.

2. Upon the receipt of a set of waypoints, each of the waypoints is converted from lat/long to northing/easting(meters).

3. Waypoint zero is set to the current location so the vehicle will move in a straight line from the start location to the first waypoint.

4. Each update from the GPS unit triggers the algorithm to check if it has reached the current waypoint, based upon the radial tolerance for that waypoint. If so, the next waypoint becomes the current waypoint, and the current waypoint becomes the last waypoint.

5. The algorithm finds the straight line between the last waypoint and the next waypoint.

6. It checks to see if it has gone past the next waypoint. If so, the geometry is reversed so that the algorithm will swing the vehicle around to hit that waypoint.

7. It calculates the error between its current heading and the heading to the point one lookahead distance, l, ahead along the straight line path.

8. Using this error, it calculates the required curvature to reach that point as $\gamma = \frac{2\Theta_{err}}{l}$.

9. The algorithm repeats this procedure for each new positional update. If it reaches the final waypoint in the path, it checks the "patrol mode" or flag status. If true, it will continue from the final waypoint to the first waypoint. If false, it halts the vehicle.

The *Pure Pursuit* algorithm [65, 6, 5, 66, 67] was devised to compute the arc necessary to return a vehicle back onto a path. It computes the curvature of an arc that a vehicle must follow to bring it from its current position to some goal position, where the goal is chosen as some point along the path to be tracked. The algorithm iterates continuously, with the goal point sliding along the path, forming a smooth tracking trajectory, as shown in Figure 16. It operates in a fashion similar to the way humans drive, fixating on a point some distance ahead on the road and attempting to follow it.

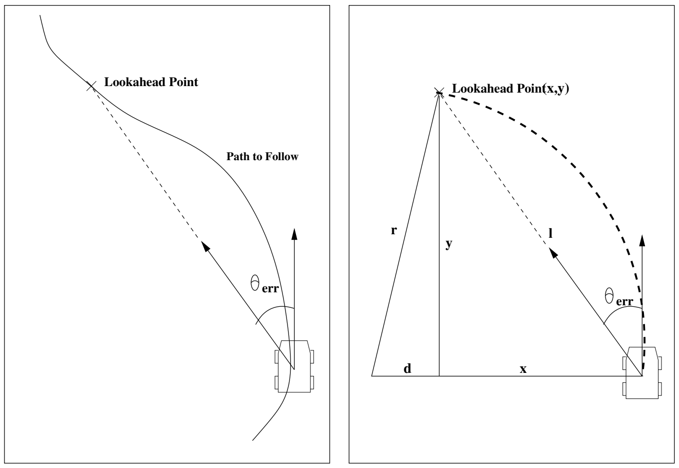


Figure 17: *Geometry of the Pure Pursuit algorithm showing the path to be tracked(left) and the calculated steering curvature(right). The curvature of the arc indicates a circle of radius* r*. Parameter* l *is the lookahead distance, with* x *and* y *defining the position of the lookahead point relative to the vehicle.*

In *Pure Pursuit*, we consider a constant curvature arc connecting the current vehicle position and a point on the path a fixed distance ahead, called the lookahead distance (l), as shown in Figure 17. The lookahead point is one lookahead distance away from the vehicle, on the path to be followed.

From Pythagoras, we have

$$x^2 + y^2 = l^2 \tag{1}$$
$$d^2 + y^2 = r^2 \tag{2}$$

In Figure 17, as r and $d + x$ are both radii of the same circle

$$d = r - x. \tag{3}$$

Substituting Equation 3 into Equation 2 yields

$$\begin{aligned}(r-x)^2 + y^2 &= r^2 \\ x^2 + y^2 &= 2rx. \end{aligned} \tag{4}$$

And substituting Equation 4 into Equation 1 yields

$$\begin{aligned} 2rx &= l^2 \\ r &= \frac{l^2}{2x}. \end{aligned} \tag{5}$$

The curvature of an arc is given as $\gamma = \frac{1}{r}$ so we can rewrite Equation 5 as

$$\gamma = \frac{2x}{l^2}. \tag{6}$$

Essentially, the *Pure Pursuit* algorithm is a proportional controller which operates on the error between the current vehicle heading and the heading to the goal point on the path. This can be seen by showing the formula for curvature in a different way. From Figure 17, we see that $sin(\theta_{err}) = \frac{x}{l}$, so for small heading errors $\theta_{err} \simeq \frac{x}{l}$. Substituting this into Equation 6, we get

$$\gamma = \frac{2\theta_{err}}{l}. \tag{7}$$

From this, we can see that the algorithm really only has a single parameter, the lookahead distance, l. This makes the algorithm exceedingly easy to implement and tune. Tuning this lookahead distance adjusts a number of performance characteristics. Having a smaller lookahead distance forces the system to track the path more accurately, and increases the maximum curvature of a path that can be tracked. Additionally, a smaller lookahead distance causes the vehicle to return to the path more aggressively when it is separated. However, there are a number of good reasons to make the lookahead distance longer:

- A longer lookahead distance reduces oscillations while tracking a path.
- For paths which have sharp turns, it allows the vehicle to begin turning before it reaches the curve, resulting in smoother trajectories.
- The commanded steering changes are less abrupt, important for vehicles with high steering lag or operating at higher speeds, as the resultant turns are less sharp.
- With a larger lookahead distance there is less overshoot when returning to the path from a large separation.

The basic *Pure Pursuit* algorithm is less stable when the vehicle finds itself a long way off the path. Normally this would not be a problem when simply tracking a path. However, the *Obstacle Avoidance* behaviour may move the vehicle away from the path a significant distance to avoid untraversable objects in the way. For this reason, the addition of an *adaptive* lookahead distance, as described by Kelly [68] provides greater stability. The adaptive algorithm differs from the standard algorithm in only one respect, the lookahead distance is now no longer a fixed length, but varies with the distance between the vehicle and the path.

For a straight line path shown in Figure 18, this distance is found as follows. $\mathbf{w}$ is the vector from the last waypoint to the current position and $\mathbf{v}$ is the vector from the last waypoint to

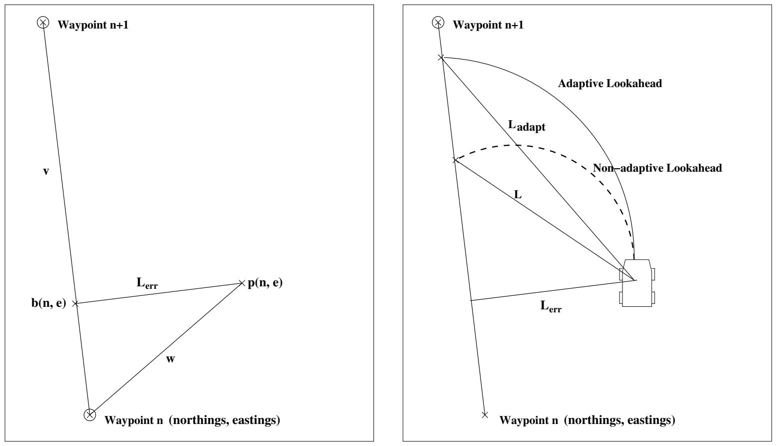


Figure 18: *Adaptive lookahead can be used to make the algorithm more stable.*

the next. Point $\mathbf{p}$ is the current position, $\mathbf{n}$ is the last waypoint, $\mathbf{b}$ is the point of projection of $\mathbf{w}$ on $\mathbf{v}$. L_{err} is the distance between the vehicle position and the path, L is the fixed lookahead distance, and L_{adapt} is the adaptive lookahead distance used by the algorithm. Subscripts n and e indicate coordinates of northings and eastings respectively.

$$\begin{aligned} \mathbf{b} &= \mathrm{Proj}_{\mathbf{v}}\mathbf{w} = \frac{\mathbf{w}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}}\mathbf{v} \\ L_{err} &= \sqrt{(p_n - (b_n + n_n))^2 + (p_e - (b_e + n_e))^2} \\ L_{adapt} &= L + L_{err} \end{aligned}$$

In order to operate in concert with other behaviours, the *Pure Pursuit* module does not directly output a vote for the ideal arc. Rather, it spreads the votes out on a Gaussian distribution around the ideal arc, so that the decision making process can select candidate arcs that will avoid obstacles, while still providing goal directed behaviour. A typical scaled vote set from the *Pure Pursuit* component is shown in Figure 19.

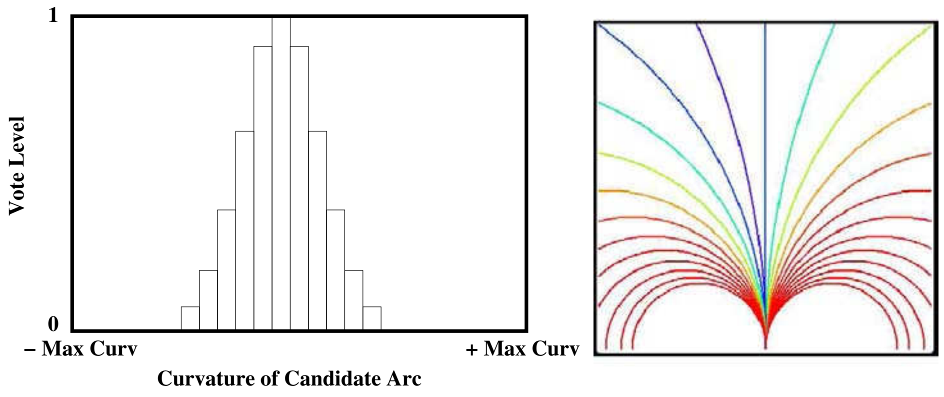
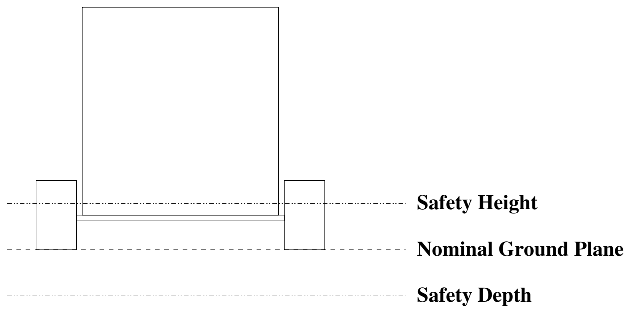
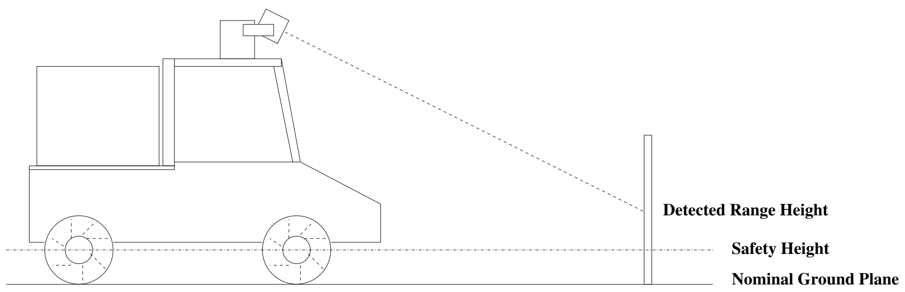


Figure 19: *A selection of candidate arcs, coloured to indicate those more desirable to follow the intended path.*

For a more detailed review of this algorithm, see [69].



(a) Back view of vehicle



(b) Side view of vehicle

Figure 20: *The Obstacle Detection module, illustrating a detected obstacle.*

5.2 Obstacle Detection

The *Obstacle Detection* module, also known as *Laser Safety*, provides a continuous safety check for UGV operation, looking for any dangerous obstacles in the vehicle's immediate path. It is a similar system to the one found in [46]. Drawing from one or more range sensors, it halts the vehicle if it finds any positive or negative obstacles (i.e. posts or holes) in the vehicle's immediate path, preventing either a tele-operator or the autonomous software from damaging the vehicle. The dimensions of the area in front of the vehicle to be checked, as well as the acceptable height and depth of obstacle, are user-definable at run-time.

The algorithm functions by checking the 3D range readings from the *Laser* component's "Range3dEvent" and adjusting for the sensor's mounted height to the bottom of the vehicle's wheels (i.e. the estimated ground plane). If it finds a user-specified number of range returns greater than a certain height tolerance above the ground plane ("Safety Height"), or lower than a certain depth below ("Safety Depth"), it will halt the vehicle by vetoing all arcs in its *ArcVote* events.

The various tolerances are shown in Figure 20(a). In Figure 20(b), the vehicle is shown approaching an obstacle that a tele-operator has failed to see in the video feed. The nodding laser rangefinder will get a range return off the obstacle at the indicated spot. The calculated height of this range reading, will be higher than the "Safety Height", and the vehicle will be halted.

The rectangular area to be checked in front of the vehicle is set by two user-defined variables, *Safety Distance*, and *Safety Width*, as is shown in Figure 21. The laser range finder will have a maximum range much greater than the area which needs to be checked for obstacles, usually 20 meters, versus 3 meters for *Safety Distance*, and any range readings which do not fall within the Safety area, are simply ignored. The user can also define at run time whether the Safety Box should be checked for positive obstacles, negative obstacles, or both.

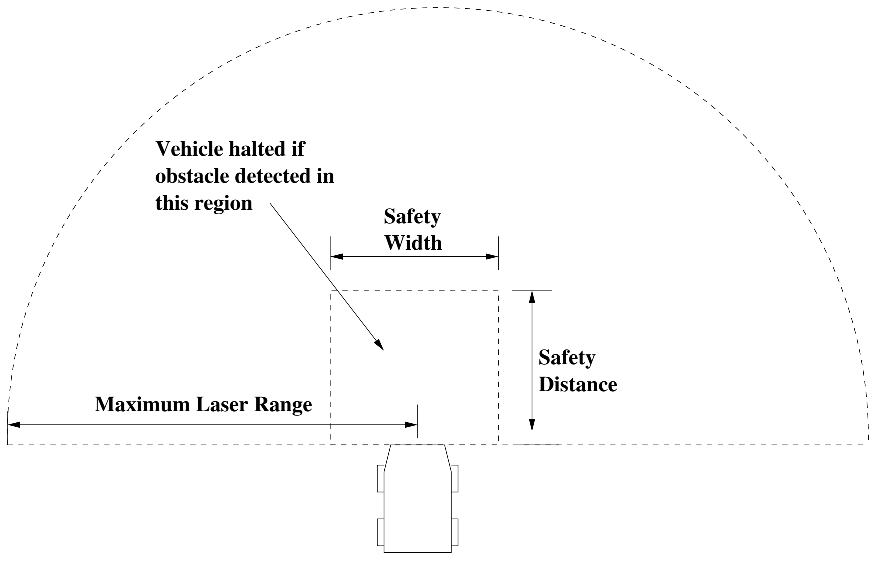


Figure 21: *The Obstacle Detection module, viewed from the top of the vehicle.*

It is important to tune the user-definable variables carefully. This algorithm is meant simply as a safety catch for other operations, and therefore, should never actually become active. If the "Safety Distance" and "Safety Width" are set too large, the vehicle will be halted before the user or the autonomous software has had a chance to avoid the obstacle. However, if the safety box is set too small, it might not be able to stop in time to not hit the obstacle. These values will of course be dependant on vehicle size, turning radius, speed, and braking distance. From experience, for the Raptor UGV driving at around 3 m/s it was found that a "Safety Distance" of 3 m was adequate.

It is also important to set the "Safety Height" and "Safety Depth" carefully. Setting them too low will result in every bump in the terrain causing a halt, whereas too high could result in vehicle damage. A general rule of thumb is to set this value to half the wheel height, or 0.3 m in the case of the Raptor UGV. A final parameter which needs to be tuned is the number of range returns above or below the thresholds which will trigger a vehicle halt. Because of the noisy nature of range returns, it is impractical to halt based on a single range return. A value of 3 was found to be practical on the Raptor vehicle. The SICK laser scanner employed has an angular resolution of 0.5 degrees. At a distance of three meters, it is theoretically possible to miss an obstacle as wide as about 7.5 cm. This however, is not a real concern, because if it is missed, it will be picked up by a subsequent laser scan (occurring every 26.6 ms).

The advantage of this *Obstacle Detection* algorithm is that it is very simple and reliable, requires very little processing power, and uses only one sensor. This sensor could also have easily been the Digiclops stereo vision camera, although this has not been tested at Defence R&D Canada – Suffield. However, this algorithm cannot provide any sort of steering directions to avoid obstacles, and can only function as a safety catch. Furthermore, because it cannot perceive an up or down slope, it may halt the vehicle when it is not necessary, which is especially problematic if the "Safety Distance" is set too far in front of the vehicle. This module is used only to complement the other UGV controls.

5.3 2D Obstacle Avoidance - Vector Field Histogram

This module provides simple, reliable avoidance of discrete obstacles during UGV operation. *Vector Field Histogram* (VFH) [70] is typically used on small robots in indoor environments to seek goals while providing reactive obstacle avoidance. It was adapted by DRDC scientists to evaluate its effectiveness on larger UGVs in outdoor environments, both in the role of standalone autonomous operation, and also in concert with other algorithms. The algorithm assumes a 2 dimensional world with discrete obstacles, and maps laser range data into a simple occupancy grid to evaluate candidate steering angles.

The actual code implemented on the Raptor UGV was taken from the Player/Stage project[71], and uses a variant of the VFH algorithm called VFH+[72]. This variation includes the dynamics of the robot when estimating which cells in the occupancy grid will be intersected by each steering angle. Specifically, the original algorithm estimated robot travel in terms of straight lines, while the *VFH+* variant estimates travel in arcs. This makes it more applicable to Ackerman steered vehicles such as the Raptor UGV.

VFH uses a two dimensional Cartesian histogram grid as a world model, which is updated continuously with range data from a forward facing SICK laser scanner. The laser was mounted with beams aiming horizontally out from the vehicle bumper. The laser was mounted low enough on the bumper that the vehicle itself could safely clear any obstacle lower than the laser's mounting height. The laser mounting is shown in Figure 22. This laser data was updated at 2 Hz, which was found to be fast enough for control given the lag time in the Raptor UGV's steering system.

Despite the 2D world assumption, the *VFH* algorithm has a number of benefits. Firstly, it relies on a simple sensor set: only one laser scanner and a rough estimate of position and heading are required. It avoids the additional complications associated with a $2\frac{1}{2}$D map, which requires accurate measurement of the pitch and roll of the vehicle platform. Secondly, it is extremely reactive, analyzing only the most recent sensor data to choose its behaviour. This makes it ideal for avoiding moving obstacles such as people or other vehicles. Thirdly, it combines an avoidance behaviour with a goal seeking behaviour in a single module. It should be noted that this algorithm has been adapted to use 3D range data in [73], but this particular implementation was not used here.

The core of the algorithm will now be described. Figure 23 shows a SICK laser scanning a world with a small and large obstacle (Obstacles "A" and "B"). First, the laser reflects

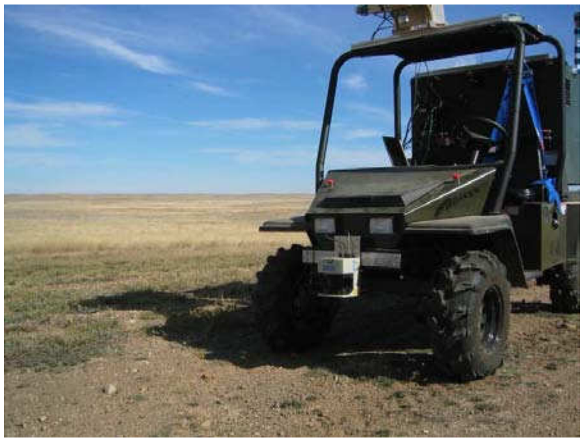


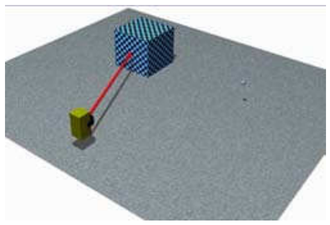
Figure 22: *The laser mounting used for the* VFH *algorithm.*

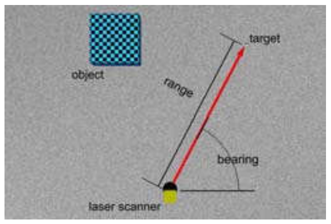
off the objects, and the range and bearing to each of the detected obstacles are calculated (Figures 23(a) and 23(b)). Secondly, these detections are marked on a grid map 23(c), typically 8 m by 16 m, with a 10 cm resolution for the DRDC implementation. This map moves with the vehicle, such that it always represents the area in front of it. When a range reading is filled into a cell, it is stored as a magnitude m, proportional to distance from the sensor. This magnitude will be zero for the furthest point in the map, and increases linearly closer to the sensor. In this way, obstacles which are closer are treated as more important than those further away. After this step, the unknown cells behind the occupied cells are filled in as obstacles in a similar way, Figure 23(d). Furthermore, the obstacle regions are enlarged by the user-specified robot width and a safety distance, to ensure that the robot does not come near these obstacles.

At this point the second stage of data reduction begins: the creation of the polar histogram. The magnitudes stored in each map cell are treated as a vector. The histogram, shown in Figure 24(a), has an arbitrary angular resolution (typically 5 degrees). Each 5 degree portion, called a sector, indicates the obstacle density as the height of the bar in the histogram. This is calculated as the sum of the magnitude vectors from the grid map for those cells which intersect the angular sector. More simply, they represent the sum of the obstacles encountered if the robot were to take that steering angle, with the vehicle paths for each steering angle modelled as an arc through the grid map.

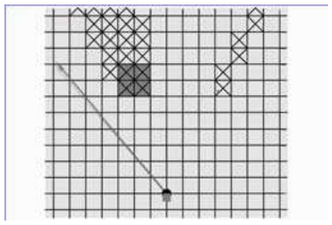
A further data reduction is now performed on the Polar Histogram, to create the Binary Histogram, Figure 24(b), using a threshold value to determine which steering angles should be considered as obstacles. Each angular sector is now considered as occupied or unoccupied: any obstacle densities which fall below the threshold are ignored and considered open.

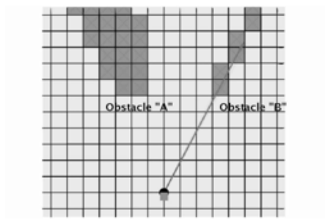
The final data reduction step converts the Binary Histogram to the Masked Histogram, Figure 24(c). Because the vehicle will have a minimum turning radius, the histogram sectors to the far left and far right may need to be blocked off to indicate that the robot is unable to turn sharp enough to avoid this obstacle. This minimum turning radius is found


(a) Laser hits obstacle



(b) Range and bearing found to obstacles


(c) Obstacles placed in occupancy grid



(d) Unknown cells behind obstacle filled

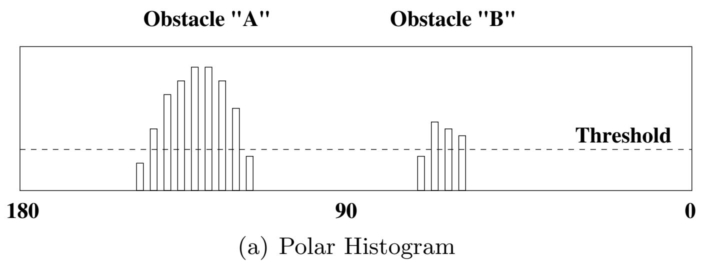
Figure 23: *The process of building an occupancy grid based on laser range data.*

from the vehicle geometry, as well as its current speed.

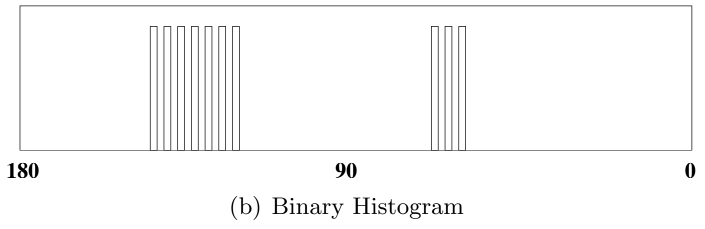
At this point, the *VFH* software can now choose a steering direction for the vehicle. The Masked Histogram shows the directions which are free and which are blocked, but depending on the goal location, some free directions are better than others. The software identifies openings as either wide or narrow. If it is narrow, the vehicle will travel through the middle of it, but if it is wide, either the left or right side will be chosen, whichever is closer to the goal. The selection of an opening is based on user-tuned parameters, or weights for both the goal direction and the current direction. This allows the robot to seek goals, but dampens any oscillatory behaviour.

The final step in the algorithm is selection of speed. This is based mostly on user-defined parameters: a maximum speed for wide openings, maximum for narrow openings, overall maximum speed, and a maximum acceleration. The values specified will depend on the vehicle and the degree of caution desired. For the Raptor UGV in cluttered environments, the maximum speed was set to 3 m/s, for wide openings 2 m/s, for narrow 1 m/s and maximum acceleration set to 0.5 m/s^2.

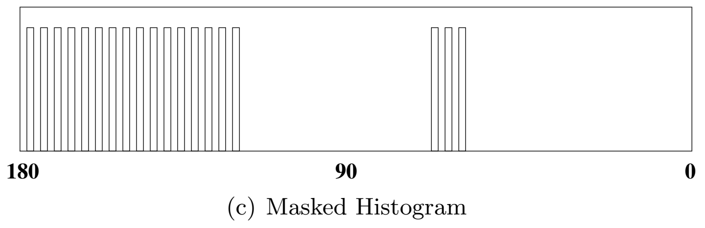
A number of modifications were made to the Player/Stage code to make it compatible with the rest of the Raptor software. Firstly, in order to retrieve position and heading information, it subscribes to the *Gps* and *Imu* services, reporting position in latitude/longitude and heading in compass degrees, respectively. These needed to be converted to meters and counter-clockwise degree units to work with the software. Additionally, the chosen arc and speed are now packed into an “ArcVote” event to send to the *Arc Arbiter*, as was described in Section 5. In order to provide the algorithm with goal waypoints from the user, it was modified to work with the “VehiclePlan” interface, as described for the Pure Pursuit



(a) Polar Histogram



(b) Binary Histogram



(c) Masked Histogram

Figure 24: *The histograms created in the VFH algorithm.*

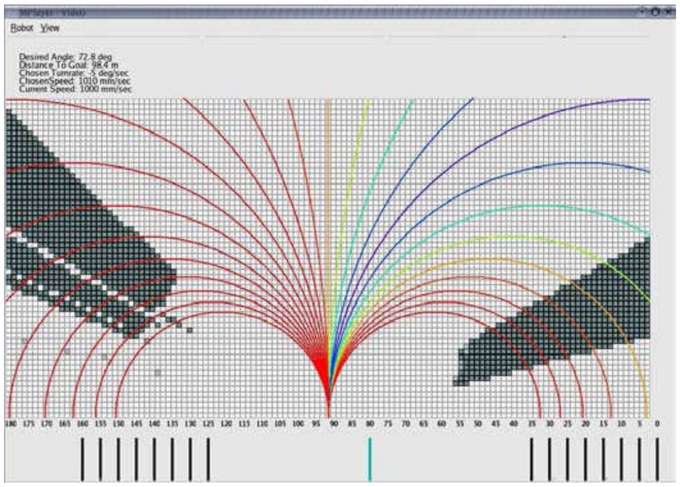
module, in Section 5.1. Finally, a graphical display was developed in the Qt language for debugging and visualization purposes. This display is shown in Figure 25. At the bottom is the Masked Histogram, which shows the obstacle sectors in black, and the chosen steering angle in blue. The vehicle is currently in an obstacle course, with the candidate steering angles shown over the map, with blue being more desirable, red being less desirable.

The run-time configurable parameters of the *VFH* algorithm are given in Appendix B.

Results of testing this algorithm are presented in Section 6.

5.4 3D Obstacle Avoidance

The algorithm described in this section is an attempt to provide reactive obstacle avoidance using 3D range data and mapping, as opposed to the more limited *Obstacle Detection* and *VFH* algorithms previously described. This component relies upon the Raptor UGV mapping software (i.e. *Terrain* and *Traverse* modules - described fully in [63]) to provide it with a description of the area directly in front of the vehicle. Similar to other candidate arc systems such as Morphin and Gestalt[57, 74], the *Obstacle Avoidance* module estimates the cost of driving candidate angles based on the *Traversability Map*, which provides a measure of the roughness, slope, and step hazards to the vehicle. The benefit of using the *Traversability Map*, as opposed to raw sensor data, is this ability to encode a variety of different types of hazards. Also, this type of representation is more readily shared between robots given limited communication bandwidth. Unfortunately using 3D data comes at the



```
Desired Angle: 72.8 deg
Distance To Goal: 98.4 m
Chosen Turnrate: -5 deg/sec
ChosenSpeed: 1010 mm/sec
Current Speed: 1000 mm/sec
```

Figure 25: *The* VFH *display, and the candidate arcs as it navigates between obstacles.*

cost of additional sensing and processing (the tilt and roll of the vehicle must be accurately measured to create a good *Terrain Map*). This module does not provide goal-directedness on its own, but instead works in concert with the *Pure Pursuit* module described in Section 5.1.

A typical traversability map and the candidate arcs are shown in Figure 26. The black arcs shown in the picture have been vetoed because of the discrete obstacles in their path, while the blue arcs have been deemed safe to travel.

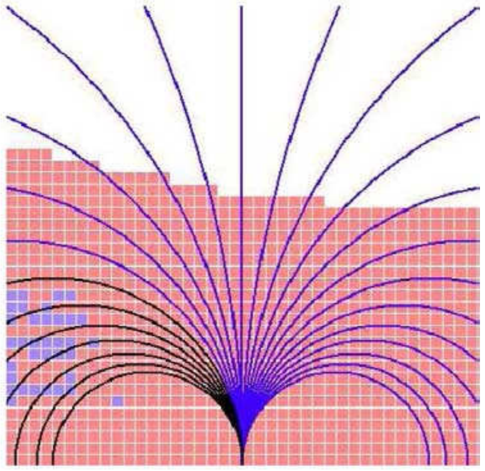


Figure 26: *A Traversability Map with overlaid candidate arcs*

The *Traversability Map*, as described in Section 4.4.0.7, contains 50 cm x 50 cm cells, which have a Goodness rating as:

- Unknown cells are assigned a goodness value of 2.
- Untraversable (Obstacle) cells are assigned a goodness value of 1.
- Traversable cells are given a goodness rating between 0 and 1 indicating the desirability of the position, based on its roughness, step height, pitch, etc.

A circular arc, $\mathbf{a}_i$, approximates the path travelled for each of the candidate steering angles, α_i. The curve's thickness approximates vehicle width by including adjacent traversability cells in the cost. Cells which intersect the ith arc form a cost vector $\mathbf{c}_i^{'}$ of size N_i. The algorithm discounts these costs through a parameter F_{disc} (user-defined between 0 and 1), the Euclidean distance to the current cell e_{curr}, and the furthest cell in the Traversability Map, e_{max}:

$$\mathbf{c}_i = \mathbf{c}_i^{'}(1 - F_{disc}e_{curr}/e_{max}) \tag{8}$$

effectively reducing the cost of distant terrain. It finds the desirability, d_i, of a given arc vote v_i, from the average of the vector of discounted costs:

$$d_i = 1 - \frac{1}{N}\sum_{j=1}^{N}\mathbf{c}_i \tag{9}$$

The obstacle avoidance algorithm vetoes arcs if any intersected cell has a cost of 1:

$$q_i = \begin{cases} 1 & : \quad \mathbf{c}_i[j] = 1 : j = 1 \rightarrow N_i \\ 0 & : \quad \text{otherwise} \end{cases} \tag{10}$$

Finally, the maximum allowed speed of each arc is based upon that arc's vote and a user-defined maximum speed s_{max} and minimum speed s_{min}:

$$s_i = d_i(s_{max} - s_{min}) + s_{min} \tag{11}$$

The vote for each of the candidate arcs, as well as the speed requested for each is then published in an "ArcVote" event to the *Arc Arbiter* for control of the vehicle.

The *Obstacle Avoidance* component's operation is determined by a number of run-time parameters. These parameters are located under the *ObsAvoid* section of the Raptor's XML configuration file. Table B.3 in Appendix B provides a complete list of these parameters and describes their significance and usage.

5.5 Deliberative Path Planning - D* Lite

A UGV tasked with navigating its way from its current location to a goal location, can simply follow the heading to the goal and reactively avoid obstacles as it encounters them, as embodied in the algorithms previously presented. This approach is simple, but can be problematic without an hierarchy of behaviours to control the UGV. For example, for a system which does not look ahead far enough, the UGV would become stuck in a cul de sac. Also, without a representation of the world, the UGV is not able to handle *a priori* knowledge. This is an important shortcoming in a military context in which a UGV should reasonably be expected to obey out-of-bounds areas.

Alternatively, autonomous vehicle navigation can be thought of in terms of an Artificial Intelligence (AI) search task. In this case, the UGV has *a priori* knowledge about its environment and incorporates this into a map of its environment. The UGV begins by searching the map for the best (shortest, least costly, etc.) path and then attempts to follow this path to the goal. As the UGV tracks the path, it continually senses its environment and incrementally adds to its map. Should it discover that its path is blocked, it then plans a modified path to the goal. This approach to autonomous vehicle navigation, having AI search as its theoretical underpinnings, is provably complete (i.e., if a path exists to the goal, it will (eventually) be found). It is also allows *a priori* information to be seamlessly incorporated into the map. DRDC's *D* Lite* component provides such an AI search task.

This *D* Lite* component provides high level path planning and goal directed behaviour based upon the *Global Map* of accumulated local *Traversability Maps.* This component is much slower to execute than the previously discussed algorithms, but has the advantage of being able to plan the robot's motion using previously acquired information, either from the robot's travels or via map data given by a user or another robot. It accounts for both desirability of terrain as well as goal direction in making its decisions. The *D* Lite* algorithm [75] combines aspects of A^* search, the classic AI heuristic search method, and incremental search to plan near-optimal paths in partially known environments. D^*Lite is reported to be at least two orders of magnitude faster than repeated A^* searches. D^*Lite is algorithmically similar to the very successful D^* [76] path planner but is much simpler to understand and thus to extend. D^*Lite is at least as efficient as D^* (in terms of the number of nodes examined in the process of finding a path to a goal) and is quicker.

D^*Lite is a graph search method, which breaks the map down into a set of nodes (i.e., grid cells in the map). The lines connecting adjacent nodes are called arcs or edges. A path is then considered to be a sequence of nodes through the map, whose cost is simply the sum of the cost of travelling each edge in the path. In order to evaluate different paths, a search algorithm begins at the start node in the map, and begins stepping through adjacent nodes, evaluating the edge costs, attempting to reach the goal node, which is the goal location in the map. In the case of this implementation, the edge costs are the goodness ratings given in the *Traversability Map*, described in Section 4.4.0.7. D^*Lite is an heuristic search method, and as such, it does not blindly search through the map nodes, evaluating all paths to find the least costly one. Rather it uses hints, in this case the Euclidian distance to the goal, to find a path more quickly.

D^*Lite is an adaptation of Koenig's Lifelong Planning A^* (LPA^*) [77] which in turn is a derivation of A^* search incorporating incremental search. Incremental search methods reuse information from previous searches to find solutions to similar problems much faster than is possible by solving each search task from scratch. Consider a goal-directed robot navigation task in unknown terrain in which a robot always observes which of its adjacent cells are traversable and then moves into one of them. From a start cell, the robot has to move to a goal cell. It always computes the shortest (or least cost in some sense) path from its current location to the goal under the assumption that cells with unexplored cells are traversable. It then follows this path until either it successfully reaches the goal or it observes an untraversable cell and is forced to recompute a shortest path from its current location to the goal. Figure 27, adapted from Koenig and Likhachev [75], shows the goal distances of all traversable cells and the shortest path from the robot's current cell to a goal cell. The top maze shows the initial path and the lower maze the situation after the robot has moved and discovered that its initial planned path is blocked. All of the cells in the map, with the exception of those adjacent to the start location, are unexplored before the robot has moved and are assumed to be traversable; these cells are painted white. Some cells, however, are known to be impassable *a priori*, hence the robot plans an initial path around them; these cells are painted black.

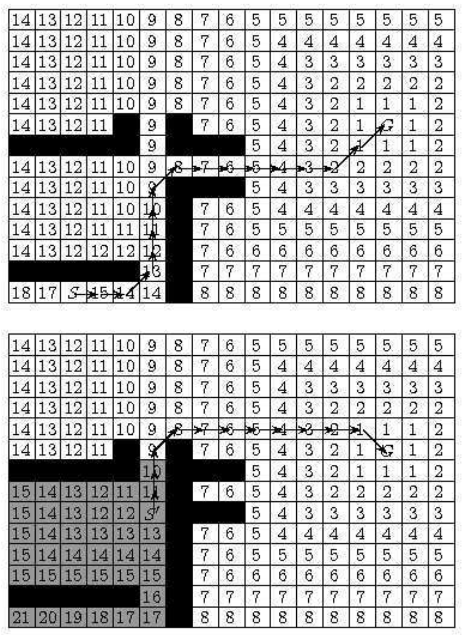


Figure 27: *Simple Maze Search*

In the top maze, the shortest path from the start location, S, to the goal location, G, is shown. This is determined by greedily decreasing the goal distance. In the lower maze, the cells whose goal distances have changed as a result of discovering that the planned path is blocked are shown in gray. Note that the majority of these changed cells are irrelevant to the re-planned path. *D*Lite* is an efficient replanner because it identifies those cells that have changed and are relevant to the replanning task.

The *D* Lite* component doesn't subscribe to any events, instead it obtains all of the information necessary to plan a path via polling the *Global Map* component. Having planned an initial path starting from the vehicle's current location and ending at the first waypoint, *D* Lite* generates "ArcVote" events at a fixed frequency attempting to track the planned path. The path planner is not deterministic in the time it takes to plan a path. If a path can not be planned in the time available, a vetoed "ArcVote" event is published. If the vehicle is unable to track the planned path because of obstacles encountered as the vehicle moves forward, *D* Lite* is forced to re-plan.

The run-time operation of *D* Lite* is configured by run-time parameters found in Appendix B. A much more detailed explanation of the background and implementation of this algorithm can be found in [78].

5.6 Decision Making - Arc Arbiter

The *Arc Arbiter* combines the output from all of the reactive and deliberative behaviours to provide coherent vehicle control. As the behaviours listed do not all operate on the same input data or guiding principles, they may have different priorities for the vehicles direction and speed. In addition, there may be any number of behaviours present at any given time. This module takes the "ArcVote" events generated by all the vehicle behaviours, and uses them to direct the vehicle controller to use the most appropriate translational and rotational velocity.

With regard to the decision making dichotomies discussed in Section 2.4, DRDC scientists have designed a completely hybrid system: there are both reactive and deliberative elements, there are some components which build a world model and some which rely directly on sensor data. The system is distributed, but relies on a single decision making entity, the *Arc Arbiter*. The individual algorithms providing input to the *Arc Arbiter* work at their own pace, but their outputs are combined asynchronously by the *Arc Arbiter* as rapidly as possible.

The *Arc Arbiter* component subscribes to the "ArcVote" events and fuses the individual behaviours into a global action using an arbitration scheme. DRDC expanded a DAMN-like [28] arbitration scheme, in which each behaviour votes for each arc in the set of candidate arcs. The behaviour gives a desirability, a certainty, a maximum speed and, if unacceptable, a veto to each arc. The IDL code listing shown below illustrates "ArcVote" event data:

```
struct ArcVoteIDL     // Defines a behaviour's vote for a single arc
{
    float curvature;  // Curvature (1/meters)
    float desire;     // Desireability: between 0 (worst) and 1 (best)
    float certainty;  // Confidence: between 0 (least) and 1 (most)
    float max_speed;  // Maximum acceptable speed (meters/second)
    boolean veto;     // "true" vetos this arc
};

// Define the available voting behaviours
enum Voting_Behaviour{HAZARDDETECT, OBSAVOID, PATHPLANNER, WAYPOINT, VFH};

// Define the number of candidate arcs in the system
```

```
const long NUM_CAND_ARCS = 25;

struct ArcVoteEventIDL                 // The CORBA ArcVote event object
{
    ArcVoteIDL VoteSet[NUM_CAND_ARCS]; // The array of votes
    TimeIDL time;                      // The time that the vote was generated
    Voting_Behaviour votingBehaviour;  // Identify voting behaviour
};
```

The *Arc Arbiter* component's event driven design gives the voting components the flexibility to run asynchronously. Further, behaviours may implement as much of the "ArcVote" object as they need. For example, the *Obstacle Detection* module uses only the veto field, while the *Obstacle Avoidance* and *Path Planning* modules use all fields. Nevertheless, the arbiter combines all available outputs into one coherent decision.

The vote based arbitration scheme and the event based delivery of "ArcVote" events creates a sliding scale of autonomy, through a subset of available components. For example, to run teleoperation control, the system needs only three modules: *Tele-operation*, *Arc Arbiter* and *Vehicle*. For autonomous operations in simple environments, *Pure Pursuit* replaces *Tele-operation*. As the environment increases in complexity, *Obstacle Detection*, *Obstacle Avoidance*, and *Path Planning* can be added incrementally to provide greater autonomous capabilities.

The *Arc Arbiter* uses votes from each available behaviour to make decisions and receives votes from each behaviour asynchronously. With each vote event, the *Arc Arbiter* asynchronously re-evaluates its decision and is, therefore, highly reactive to incoming data.

More formally, the "ArcVote" is defined as follows: the interface passes a vote for each of a set of candidate steering arcs distributed evenly between the maximum left and right steering angles, as shown in Figure 26. The vote for each of the candidate arcs is $\mathbf{v}_i \equiv \langle \alpha, d, p, q, s_{max} \rangle$. Where α is the curvature of each steering arc, $d \in [0, 1]$ is the desirability (the vote), $p \in [0, 1]$ is the certainty of this decision, $q = 0 \mid 1$ is a veto, allowing any behaviour to disallow a steering command, and finally s_{max} is the maximum allowable speed the vehicle may travel along this arc.

The *Arc Arbiter* chooses the candidate arc with the highest combined desireability and certainty, setting the speed to the arc's lowest s_{max} and disallowing any vetoed arcs. If all arcs are vetoed, the vehicle is stopped. The algorithm selects an arc by compiling the votes into a combined vote set $\mathbf{v}_c$, composed of the combined desirabilities d_c, and the combined maximum speeds s_c. For the ith arc, the combined vote is $\mathbf{v}_{ci} =< d_{ci}, s_{ci} >$. There are a certain number of behaviours, b, present, each with a weight of importance, w. The calculation consists of:

$$d_{ci} = \begin{cases} 0 & : \quad q_j = 1 : j = 1 \to b \\ \sum_{j=0}^{b} d_j p_j w_j & : \quad \text{otherwise} \end{cases} \tag{12}$$

$$s_{ci} = \min(s_j) : j = 1 \to b \tag{13}$$

This means that the algorithm steps through all of the behaviours for each candidate arc (from $j = 0$ to $j = b$). If the arc is vetoed ($q_j = 1$), its combined desirability, d_{ci}, is set to zero. Otherwise, the desirability is set to the sum of each behaviour's vote multiplied by its certainty and that behaviours weight. Each behaviour's vote is discounted by the certainty of its decision, and also by the weight value that the arc arbiter places on its votes. For example, the weight for an obstacle avoidance behaviour is typically set higher than that for a goal seeking behaviour, to indicate the relative importance. Finally, the maximum speed for that arc, s_{ci} is set to the lowest value that any of the behaviours contributed. This entire process is repeated for each of the i candidate arcs (typically 25 of them).

At this point, the ArcArbiter chooses the final arc to be sent to the vehicle (α_f, s_f) from $\mathbf{v}_c$:

$$\alpha_f = \alpha_i, \quad s_f = s_i \tag{14}$$

where i is the candidate arc with the highest value d_c. If $d_c = 0$ for all i, then the vehicle is stopped.

The output of the above process is the curvature of the winning candidate arc for the vehicle to travel, $\gamma = \alpha_f$, and the velocity setpoint $v_{\text{set}} = s_f$, as chosen above. The Arc Arbiter converts this curvature to a rotational and translational velocity for the vehicle to execute. The required rotational velocity, ω, is calculated from the curvature, γ, and the velocity setpoint, v_{set},

$$\omega \quad = \quad \gamma \times v_{\text{set}}$$

These values for ω and v_{set} are then passed to the vehicle controller.

The main difference between this method and the DAMN arbitration scheme is the addition of the certainty value p_j. This value gives each behaviour a say in its own weighting scheme via the certainty value. If a behaviour knows it is relying on poor sensor data, or if the decision it found was not critical, it may reduce the certainty value to indicate this to the *ArcArbiter*.

A few other features were added to the *Arc Arbiter* as well. Firstly, a timer was run, typically once a second to ensure that the vehicle was being controlled by at least one behaviour (i.e. that the *Arc Arbiter* has received votes). Secondly, the Arbiter also checks for stale votes: if a behaviour has not voted for a period of one second, its votes no longer contribute to the decision. Finally, the *Arc Arbiter* checks that at least one goal directed behaviour was running the vehicle (such as Pure Pursuit, D* Lite or VFH) so that the vehicle does not wander aimlessly.

6 Results

Results obtained from each of the described navigation algorithms are presented in the following sections.

6.1 Pure Pursuit

Using the Raptor platform discussed in Section 3, a number of tests were undertaken to verify the performance of the Pure Pursuit algorithm. All of the tests were done with the algorithm following the straight line between high level waypoints, as described in Section 5.1. Some test results are presented below. When evaluating these tests, it is important to note that the only navigation sensor used was GPS position and heading. This means that positional data arrives discontinuously, and is prone to sudden jumps in value. In addition, it was observed that the heading information was inaccurate, prone to large fluctuations and lagged actual heading during turning. Despite these handicaps, the algorithm as implemented performed admirably.

6.1.1 Lookahead Distance

The first set of tests illustrate the effect of changing the lookahead distance on the algorithm. For this test, the last and next waypoints were given to the algorithm, defining a straight line for the vehicle to follow. The Raptor vehicle was then started at a distance offset from this line, to illustrate the step response of the algorithm.

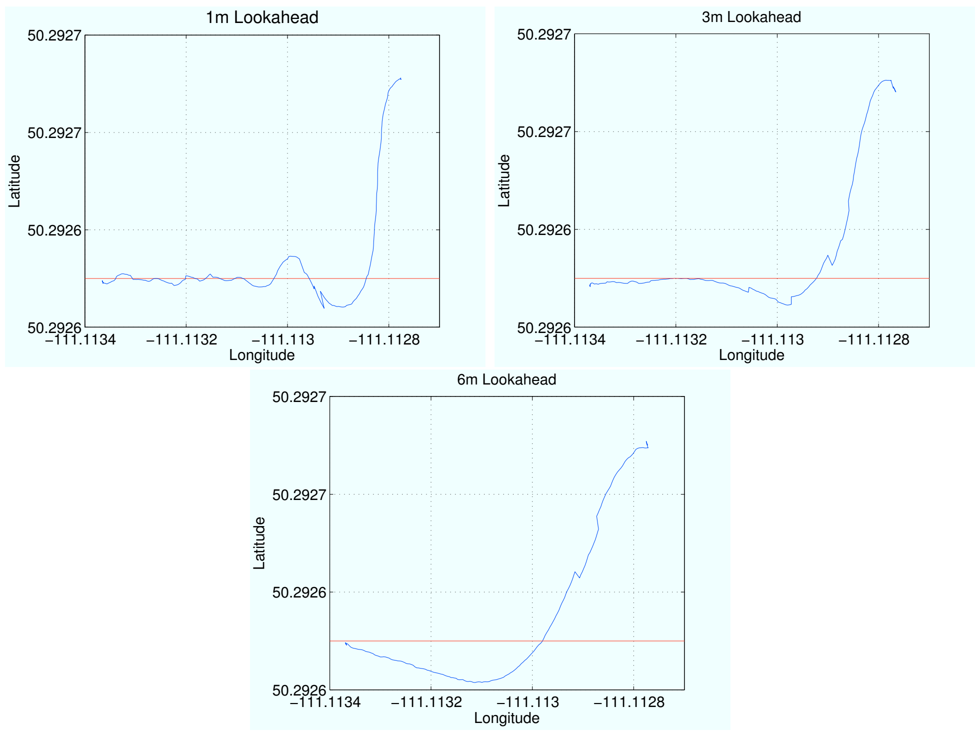


Figure 28: *The effect of lookahead distance on performance when returning to a straight path (L = 1, 3 and 6 meters).*

Figure 28 shows the position of the vehicle in Lat/Lon coordinates. The vehicle begins at the right side of the graph and moves towards the left. The red line across the lower portion of the graph indicates the line it is attempting to follow. For these tests, the lookahead distance (non-adaptive) was set to 1 m, 3 m and 6 m respectively. It can be seen that a shorter lookahead distance forces the algorithm to pursue the path more aggressively with less cumulative error, but results in sharper steering commands, and oscillation while following the path.

6.1.2 Adaptive Lookahead

The second test involved adding the adaptive lookahead parameter, as described in Section 5.1. The test setup is the same as in the first, except that for both graphs, a lookahead distance of 1 m is used. It can be seen in Figure 29 that when the adaptive lookahead distance is used in the second graph, much better performance results. At the start the path is not pursued as aggressively with adaptive lookahead, but it is much smoother. By comparison with the non-adaptive lookahead results in Figure 28, the resultant trajectory can be made much smoother without sacrificing tracking accuracy.

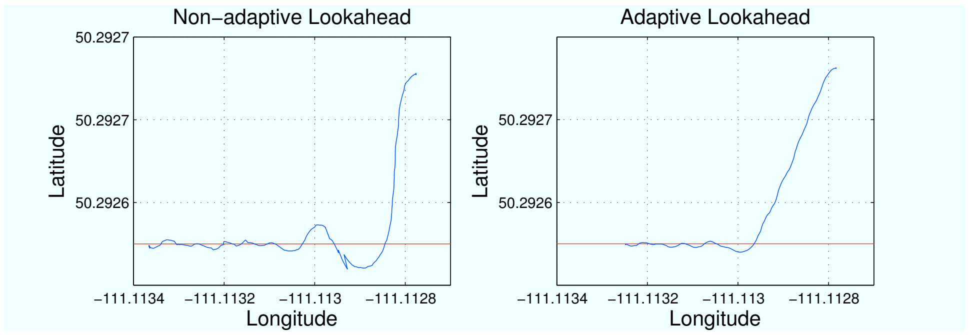


Figure 29: *The effect of using adaptive lookahead to create more stable control (L = 1m).*

6.1.3 Radial Tolerance of Waypoints

Another adjustable parameter unique to this implementation is the radial tolerance assigned to each waypoint, the distance away from the waypoint at which the robot considers that it has reached it. The problem of *subgoal obsession* is well known in robotics, where it is desirable for the robot to follow the path accurately, but not desirable for it to spend a large amount of time pursuing an individual subgoal. If the radial tolerance is set too large, the robot will move on to the next waypoint without really sticking to the path. However, if the radial tolerance is too small, then an Ackerman steered robot will overshoot the path where sharp corners occur at the waypoint. Additionally, with a small radial tolerance, if the robot has been drawn off the path by the *Obstacle Avoidance* behaviour, there is more chance of the robot unecessarily swinging around in a large arc to hit a waypoint. In order to test the effect of radial tolerance, the Raptor platform was given a tight rectangular path to follow. This is somewhat difficult for the Raptor, as the PID loops in the steering controller cause a time lag in the execution of a steering angle causing path overshoot. Additionally, the platform has a rather large minimum turning radius of around 4 m. The results for a 1 m and 6 m radial tolerance are shown in Figure 30 (lookahead distance was set to 3 m). In the graph, when the radial tolerance is set too low (1 m) for the vehicle, the path is overshot at the corners. With a more realistic radial tolerance, the robot does not approach the waypoint itself as closely, but is able to adhere to the path more accurately.

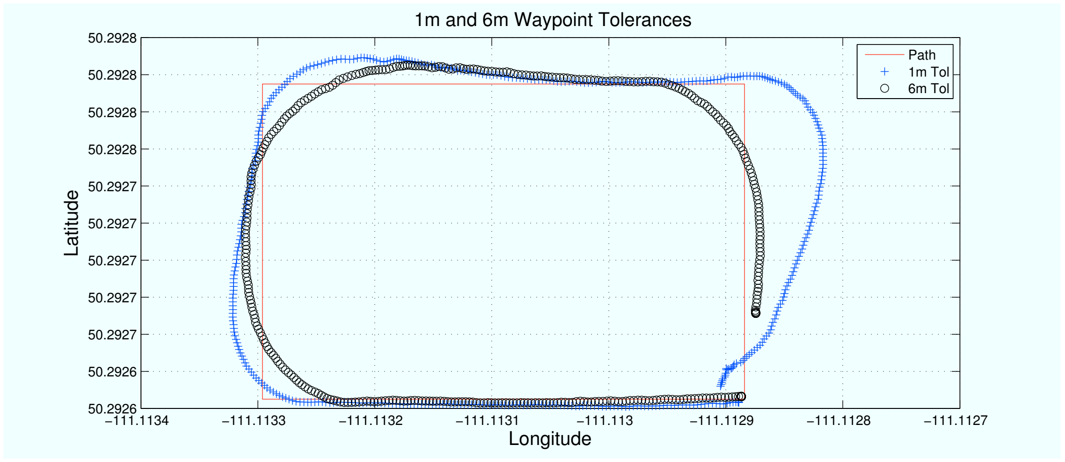


Figure 30: *The effect of changing the Waypoint Tolerance*

6.1.4 Leader/Follower Behaviour

In this experiment, the human leader drove a number of amorphous paths in the test area, while GPS breadcrumbs were relayed to the autonomous follower vehicle every few seconds. This system is a simplistic approach, with the follower simply pursuing the straight line between itself and the last leader position. Results of one test of the leader/follower behaviour are shown in Figure 31. The system seemed to perform adequately, but it is evident that such a simplistic approach causes the vehicle to cut corners, as it is not attempting to follow the exact path of the leader, but only drives towards the spot where the leader relayed positional data.

6.1.5 Waypoint Following

The final test presents the algorithm's reliability in following a high level, long distance path provided by a user control station for a patrol mission. The vehicle was tasked to drive down a dirt path for approximately 500 m, and then patrol a large loop area (Figure 32). Total path length was approximately 2.5 km. The algorithm proved reliable over this long distance, and had no problem staying directly in the middle of the straight dirt path section. From experimentation for best performance, waypoints were spaced at 10 meters, and lookahead distance and waypoint tolerance were 3 m and 5 m respectively.

6.2 Vector Field Histogram

The *VFH* algorithm was tested in the area surrounding the Bldg 34 complex at Defence R&D Canada – Suffield, shown in Figures 34, 35, and 40. This area is not particularly challenging to a UGV, but was chosen because of the assumptions inherent with using the

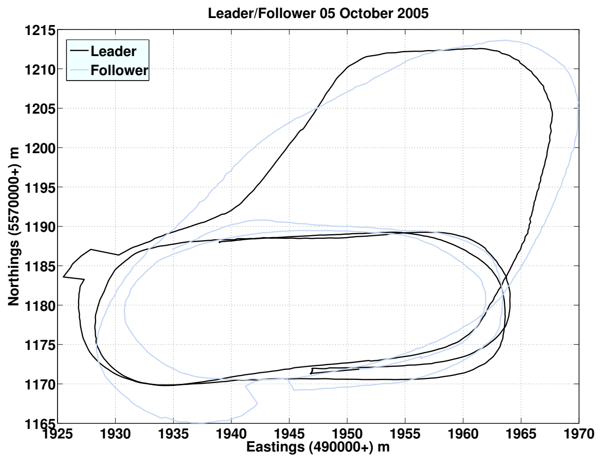


Figure 31: *Paths of two vehicles, one following the other. Note the man driven vehicle trajectory appears smoother than the autonomous vehicle — evidence of straightline trajectories used between waypoints.*

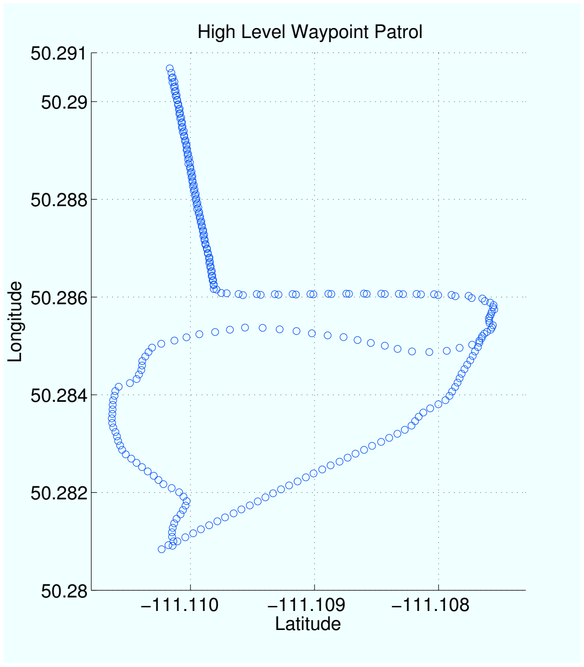


Figure 32: *Path taken by the autonomous Raptor on a 2.5km waypoint patrol. The start section down the straight dirt path is at the top.*

VFH algorithm, namely a lack of negative obstacles (holes and ditches). The trials were comprised of three distinct phases:

- An open, built up area where the vehicle would need to navigate around buildings, vehicles, etc. to reach a goal position.

- A dense obstacle course consisting of pylons arranged in a specific pattern, designed to test the algorithm's ability to navigate the vehicle in tight quarters.
- Dynamic obstacles, such as moving people, in order to test the reactivity of the algorithm.

6.2.1 Open Built Up Area

In this test, the Raptor vehicle was started on one side of the Building 34 area, and given a goal location on the other side, as shown in Figure 33. The start and goal locations were intentionally positioned such that the robot would need to dodge two large buildings as well as pipes, sheds, other vehicles, etc. on its way to the goal.

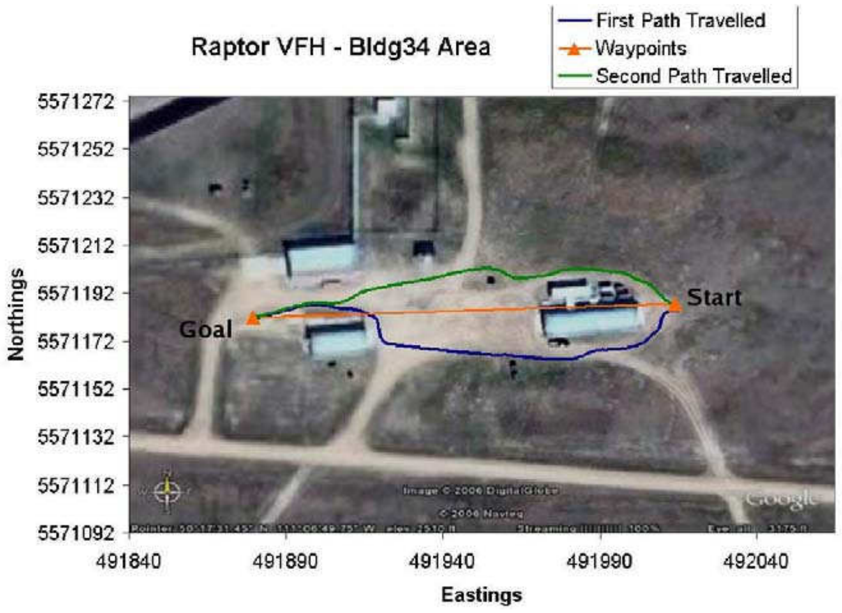


Figure 33: *The waypoints given, and the path travelled by the robotic vehicle.*

Figure 33 shows the results of the trials. The start and goal positions are indicated in orange. The first path taken is indicated in green, with the second in blue. The robot decided to take two different paths around the building due to a slight change in the start position of the vehicle. The type of terrain around the buildings is shown in Figure 34. The robot moved with an average speed of about 3 m/s during these trials, and had little problem with the obstacles encountered.

6.2.2 Obstacle Course

The second set of trials was much more difficult. An obstacle course consiting of a pre-arranged pattern of pylons was constructed to determine how well the *VFH* algorithm could navigate in a cluttered area. The course is shown in Figures 35 and 36. The robot would take a different route through the course, depending on the exact initial starting position,

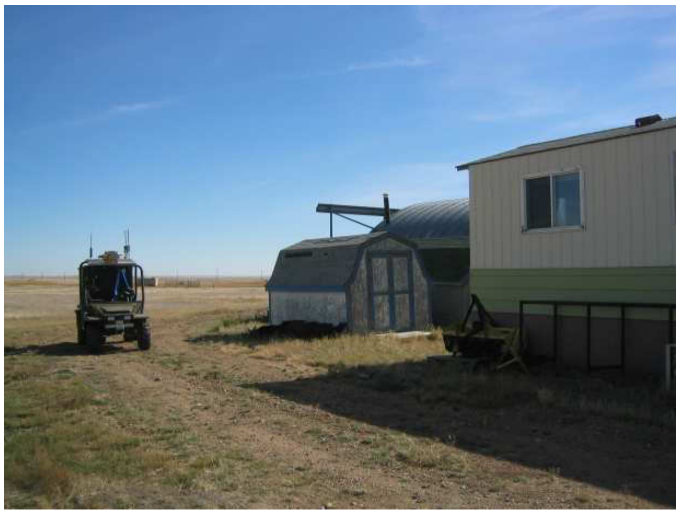


Figure 34: *The Raptor during its traverse around the Bldg. 34 area using the VFH algorithm.*

but a typical run is shown in Figure 36, averaging around 1.5 m/s without contacting any pylons.

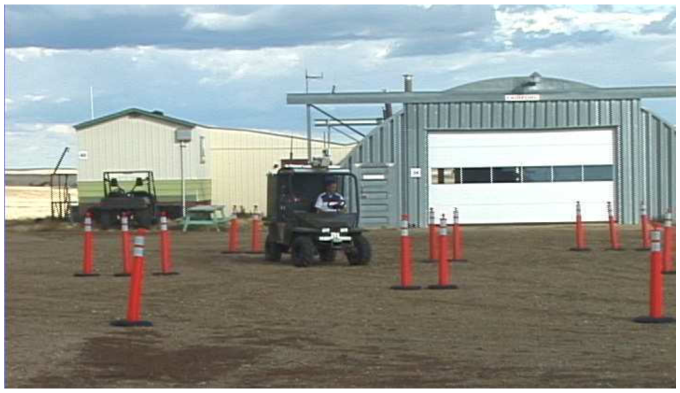


Figure 35: *The course used to test the VFH algorithm among dense obstacles.*

The display in the bottom right of Figure 37 shows the *VFH* grid map with obstacles for six pylons. You will notice that the areas behind the pylons are also filled in as obstacles, resulting in each pylon having a “ray” appearance. At the bottom of the display is the Masked Histogram, which shows the obstacle sectors in black, and the chosen steering angle in blue. The vehicle is currently in the obstacle course at the positions as shown in the top and left hand portions of Figure 37. The robot is at the location given by the blue arrow.

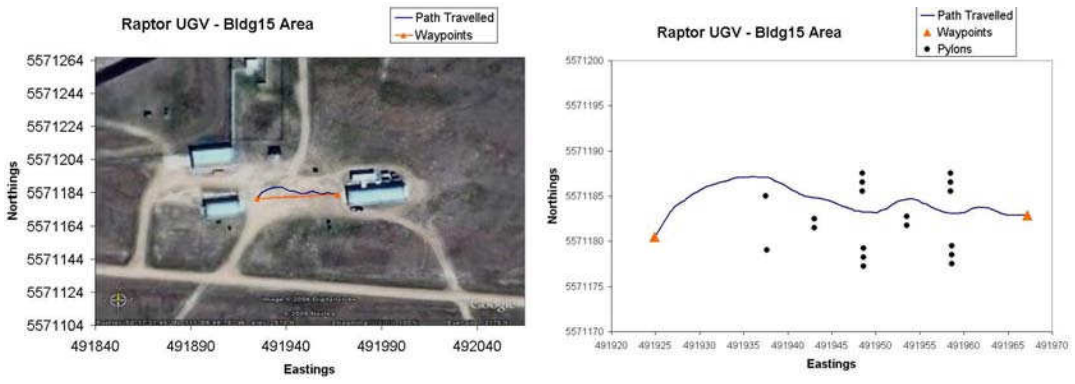


Figure 36: *The course used to test the VFH algorithm among dense obstacles.*

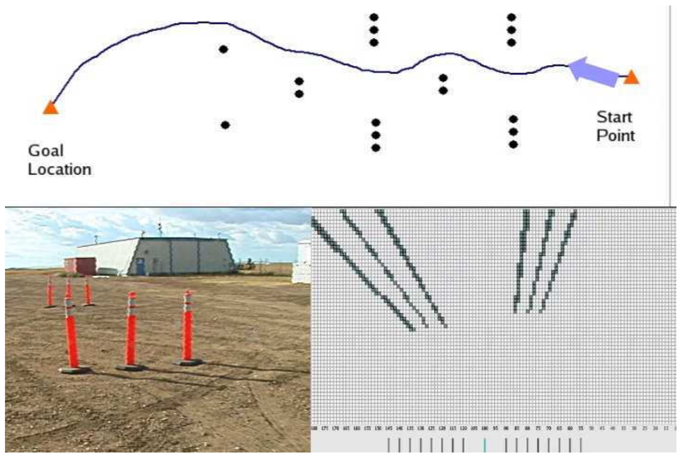


Figure 37: *A typical obstacle course run with the VFH display.*

6.2.3 Dynamic Obstacles

No specific results are documented of the robot using the *VFH* algorithm interacting with dynamic obstacles. However, from the researcher's experience, it would be nearly impossible to get the Raptor vehicle to hit a walking person, with the vehicle moving at 3 m/s. This was true of both a head on approach, or an intersect approach. The algorithm would first attempt a dodge maneuver to avoid the human, and if the human was quick enough to adjust and get within the "Safety Distance" of the robot, it would immediately be halted. With a running human, it was still very difficult to contact the vehicle, without it being halted by the *VFH* algorithm first. This algorithm proved to be extremely reliable and robust.

The one problem with this algorithm was the difficulty presented by sparse vegetation. At long distances, grasses such as shown in Figure 38 would not show up as obstacles in the map. Once the vehicle got near to the grass, however, the vegetation would appear as obstacles, and the vehicle would become stuck. For this reason, and the lack of detection of negative obstacles, it is apparent that the *VFH* algorithm is not suited for fully autonomous operations, but rather for those situations where a user would be able to assist the vehicle from time to time. With the addition of the *Obstacle Detection* module, however, the vehicle is prevented from doing anything dangerous until the user can intervene.

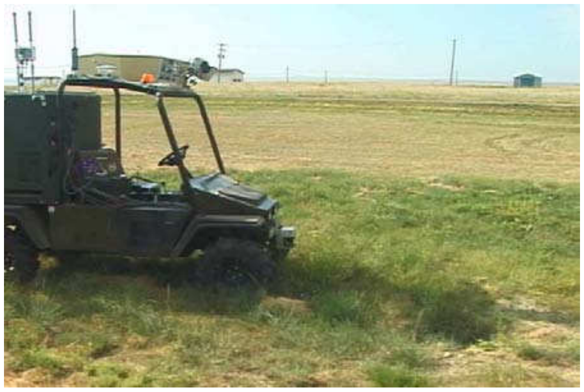


Figure 38: *Typical loose grass which would give the VFH algorithm trouble.*

6.3 3D Obstacle Avoidance

Having already successfully demonstrated simple obstacle avoidance using the system the previous fall [79], all efforts were directed towards improving the accuracy and robustness of the mapping software. Therefore, trials were conducted to evaluate the effectiveness of *Traversability Mapping* and *Obstacle Avoidance* modules for long range point to point navigation.

The UGV was given a series of GPS waypoints shown in Figure 39. As previously mentioned, a *Traversability Map* was generated using the process described in Section 4.4.0.7. This *Traversability Map* was used as input to the *Obstacle Avoidance* algorithm to detect and avoid obstacles, while the *Pure Pursuit* module evaluated each arc based on its ability to steer the UGV towards the straight line path between waypoints. The *ArcArbiter*, selected the best arc from the competing votes.

The *Traversability Map* was configured with 50 cm x 50 cm cells, while the size of the map was 16 m x 16 m. The maximum speed the UGV traveled was roughly 1 m/s. This was limited partially by the nodding rate of the lasers. Due to an issue with the stepper motors the lasers weren't able to nod faster than roughly 25 degrees/s, as this would cause the stepper motor to jitter and produce noisy data.

Step and slope hazard parameters where chosen as 25 cm and 15 degrees respectively. The low slope and step thresholds were chosen such that even slight positive/negative obstacles such as road grades and ditches would appear as hazards. Terrain representative of the course can be seen in Figure 40.

The UGV was given a series of 5 waypoints to follow, as seen in Figure 39. These waypoints

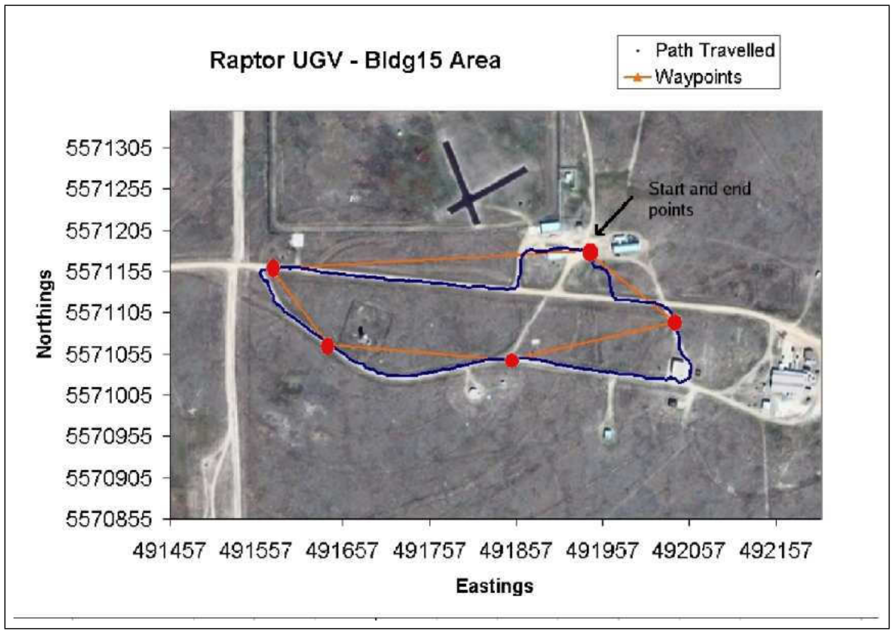


Figure 39: *Site of the Raptor autonomous road following field trials. The red points indicate the GPS waypoints which the vehicle must navigate. The orange track indicates the straight line trajectory which the UGV attempts to follow, while the blue track indicates the actual trajectory which the UGV followed as a result of obstacle avoidance.*

were located at various positions on a grid road bounded by a graded ditch and grassy areas. The UGV was able to navigate through all 5 waypoints with minimal intervention [1]. As can be seen in Figure 39, the actual trajectory taken by the UGV consistently followed the road. This proves that the slope and step hazard metrics are sufficient for detecting negative obstacles. In addition, two buildings (positive obstacles) were successfully avoided.

As the UGV was not explicitly road following, it would sometimes exhibit undesirable behaviour which nonetheless was consistent with the algorithms employed. For instance, where the grade of the ditch was relatively flat (below the slope threshold) the UGV would travel through the ditch following a trajectory to the nearest waypoint. As the grade of the ditch steepened, the UGV would navigate back to the road as the ditch was once again classified as impassable. At one point during the run the UGV turned onto an approach and subsequently got stuck instead of continuing to follow the road. This suggests that a road recognition algorithm could be used in conjunction with the Traversability Map in order to explicitly label areas of the Traversability Map as road/non road.

Due to the *Terrain Map's* statistical nature, the derived *Traversability Map* sometimes detected false obstacles. This could cause the UGV to turn erratically or stop (if the obstacle appeared near the vehicle). Once the *Terrain Map* accumulated more sensor data, yielding a more accurate statistical representation, these obstacles usually disappeared. However, in

[1]The UGV turned off the road twice in an effort to take a more direct approach to a waypoint. In addition, the UGV was halted once as vegetation was incorrectly classified as impassable

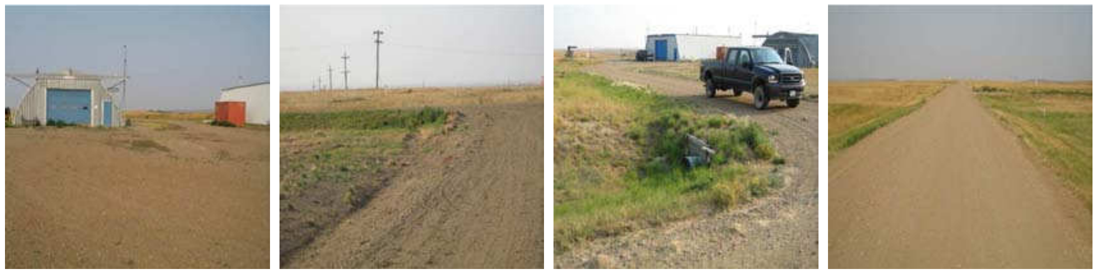


Figure 40: *Terrain representative of that found on the trial site. (Far-Left) start area - The UGV would have to navigate around the building in the left side of the image. (Left-Center) steep ditches characteristic of the trial site. These were seen as obstacles by the UGV. (Right-Center) - harsh terrain which the UGV had to negotiate (Far-Right) - straight away bounded by sloped ditches on either side*

early testing it was discovered that the vehicle could get caught if a false obstacle appeared close to the UGV, and then the vehicle moved so the laser could no longer see this position. In these instances no new range data was collected for the corresponding cell. In order to alleviate this problem, functionality was added which cleared the *Terrain Map* data, and accumulated new data for a number of seconds before resuming normal operation.

6.4 D* Lite

In an effort to gauge the performance of the D^*Lite algorithm in solving the autonomous vehicle navigation problem in unknown terrain, a number of simulations were conducted in a rectangular 2D grid world. These included examinations of the effect of the size of the world and the fraction of free space in the world. The performance of the D^*Lite algorithm is compared to that of repeated A^* searches.

6.4.1 Grid World

The grid world is eight-connected, the robot can move from its current cell forward or backward, left or right, or diagonally into an adjacent free cell. If a neighbour cell can not be reached, the cost of traversal to that cell is 10; otherwise, the cost of traversal is ≤ 10, in fact for these tests, it is 1. It may be known *a priori* that a neighour cell can not be reached, i.e., the neighbour cell in question is an obstacle, or it may be discovered to be an obstacle as the robot approaches it. To provide this capability, each cell of the grid world maintains two arrays of traversal costs. The first contains the *a priori* cost of traversal that would be incurred by the robot in moving from this cell to each of its neighbour cells. The second contains the costs of traversal that can be *sensed*. As the robot moves, it *senses* the costs of traversal to the cells within its sensing radius of its current location by querying those cells for their sensed costs of traversal. For all of the simulations described here, the sensing radius was set to its default value of 1.

To construct a grid world instance, an array of traversability values is generated automatically in a pseudorandom fashion while maintaining a specific fraction of traversable cells. Two arrays are actually used; the first, the plan array, contains *a priori* traversability information and the second, the sense array, will contain modified traversability information that the robot can sense as it moves in the world. Initially, the second is just a copy of the first. The sense array is edited manually to provide additional obstacles that can be discovered as the robot moves. The plan and sense arrays are then used to generate each cell's cost of traversal arrays, as described above. Each time the simulation is started, the grid world is constructed anew from the plan and sense arrays.

The start and goal locations for these tests were set to the lower left cell and the upper right cell of the grid world, respectively. For a traversable fraction of $\leq 50\%$, the likelihood of generating a world in which no path exists between the start and goal locations is significant. Grid worlds in which this was the case were edited manually to provide a clear path in proximity to the start and goal locations.

Discovered obstacles are provided in a two-step process. First, an initial path was planned between start and goal. Then a location on the path was chosen and manually marked as a discovered obstacle. When the planner is rerun, the robot will discover that its initial path is blocked and will be forced to replan.

6.4.2 Performance Metrics

Both A^* and D^*Lite maintain a priority queue with the most promising node for expansion always at the front of the queue. In this implementation, the underlying representation of the priority queue is a heap. Although this is an efficient representation, heap operations are nonetheless expensive. For A^* and similar searches that maintain a priority queue, one can reasonably expect that the number of node expansions (equivalent to the number of insertions and hence reorderings of the priority queue) would be a good metric for comparison of algorithm performance. As well, the number of node allocations also gives some indication of the level of effort required to obtain a solution to a search problem and thus provides another useful metric for comparison of algorithm performance. In the tests described in the following sections, the number of node expansions and the total number of node allocations required by D^*Lite and repeated A^* searches are compared.

6.4.3 Effect of Traversable Fraction

To examine the effect of the traversable fraction on path planning algorithm performance, 100x100 grid worlds were constructed with three traversable fractions: 50%, 60%, and 70%. Five replicates were generated for each traversable fraction. For each grid world, a single discovered obstacle was provided to force replanning using the method described in section 6.4.1. With repeated A^* search, an initial path was planned, then the planner was restarted from the point at which the obstacle was discovered. The number of node expansions and allocations is the sum of the values from both the initial and the replanning searches. The D^*Lite planner was simply run from the start location to the goal, allowed to discover the obstacle and forced to replan. The number of node expansions and allocations for the D^*Lite planner include the replanning episode, but arise from a single execution of the planner. In Figure 41, the number of node expansions and node allocations in a search in a 100x100 grid world with a single discovered obstacle are plotted for three traversable fractions: 50%, 60%, and 70%. Each datum is the average of the five replicates and the error bar represents ±1 sample standard deviation.

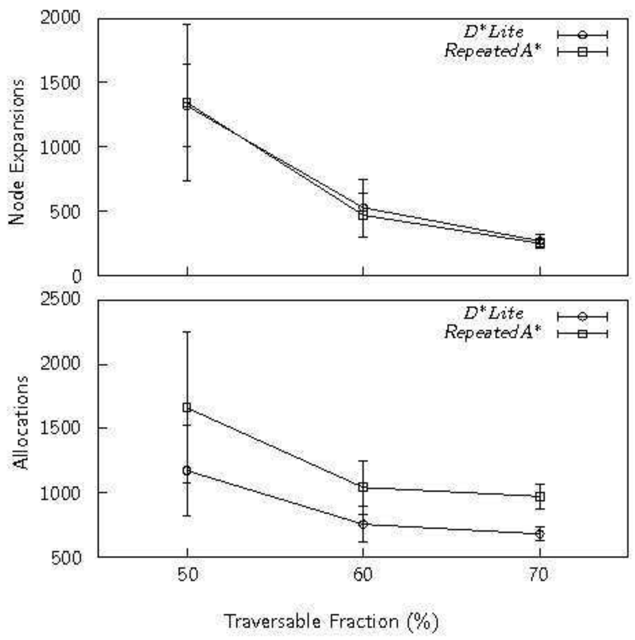


Figure 41: *Effect of traversable fraction on the number of node expansions and allocations for a 100x100 grid world.*

As the traversable fraction increases, both the number of nodes expanded and allocated decreases. This simply reflects the greater ease with which a path can be planned when less of the terrain is impassable. With a 100x100 grid world and only a single discovered obstacle, there appears to be little difference in the performance of the D^*Lite and repeated A^* searches, measured with these performance metrics. The number of node expansions required is essentially the same for both D^*Lite and repeated A^*. The only significant difference is the higher number of node allocations required by repeated A^* in comparison to D^*Lite. Again, this is to be expected since D^*Lite and A^* plan exactly the same initial path. After encountering the discovered obstacle, D^*Lite retains all of the nodes allocated during the initial search, and, in particular, the nodes on the segment of the initial planned path lying beyond the discovered obstruction. During the replanning episode, D^*Lite only need expand enough nodes to find a traversable connection between the robot's position at the point it discovered that the path was obstructed and a point beyond the obstruction on the minimal cost tree rooted at the goal node. Whereas the planner using a repeated A^* search during replanning must find a completely new path from the point at which the obstacle is detected. Thus with repeated A^* searches many nodes are reallocated during a replanning episode.

6.4.4 Effect of Grid World Size

The grid world used in the previous section to examine the effect of the traversable fraction was quite small, only 100x100 cells. To determine the effect of grid size on algorithm performance, path planning searches were conducted on three square grids: 100x100, 200x200, and 500x500. As in the traversability fraction simulations, five replicates were used. In this case, however, a path was planned in each replicate grid world, first with a single discovered obstacle and then repeated with an additional discovered obstacle. The results of these tests are shown in Figure 42. With an increasing grid size, both the number of node

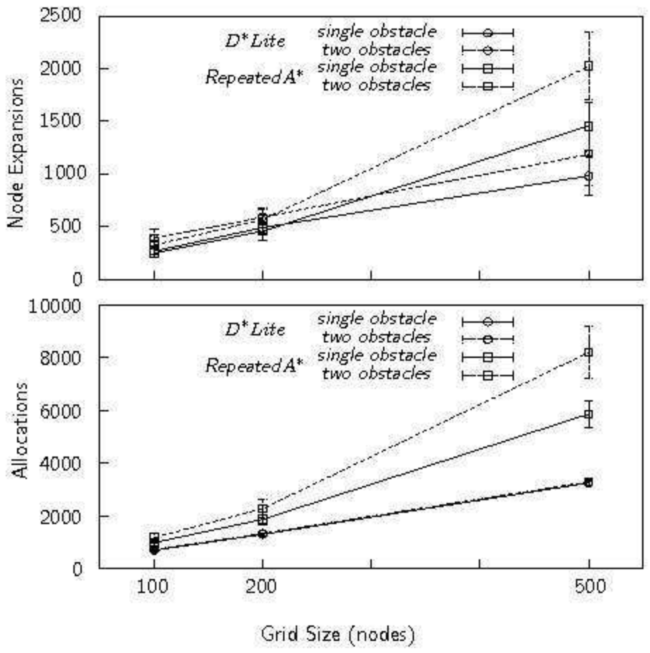


Figure 42: *Effect of grid size on the number of node expansions and allocations.*

expansions and the number of allocations during the search are increased. This is the case for both D^*Lite and repeated A^* searches. The numbers of node expansions and allocations required in repeated A^* searches in a 500x500 grid world are both clearly significantly higher than those required by D^*Lite. With the addition of a second discovered obstacle, this effect is even more pronounced. However, with the addition of a second obstacle the numbers of node expansions and allocations required in the D^*Lite search appeared not to change significantly.

6.4.5 Multiple Discovered Obstacles

To examine the effect of multiple discovered obstacles on the algorithm performance in greater detail, path planning searches were conducted on 500x500 grid worlds with up to four discovered obstacles. As with the other tests, five replicates of each grid world were used. The number of node expansions and allocations for zero through four discovered obstacles are shown in Figure 43.

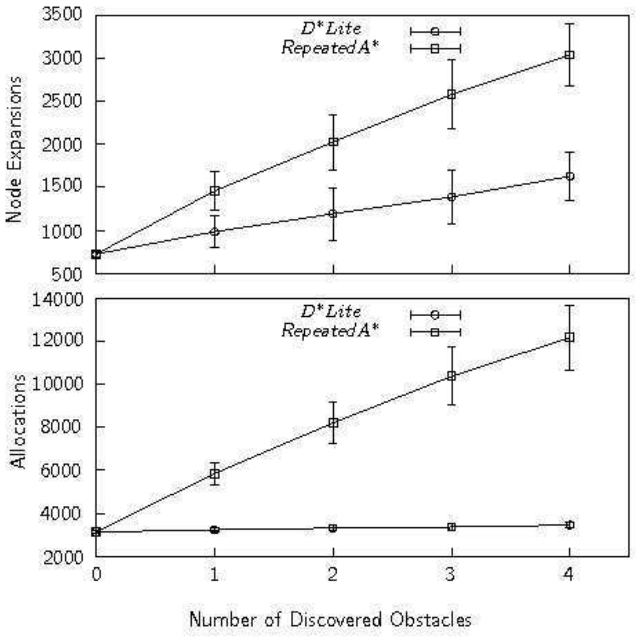


Figure 43: *Effect of the number of obstacles discovered during search in a 500x500 grid world on the number of node expansions and allocations.*

The requisite number of node expansions increased with the number of discovered obstacles in a linear fashion for both repeated A^* and D^*Lite searches, however, the rate of increase observed with repeated A^* searches was twice that observed with D^*Lite. The number of node allocations increased with the number of discovered obstacles with repeated A^* searches as expected from the discussion in section 6.4.3. In contrast, the number of node allocations required with D^*Lite appeared to be almost constant.

7 Conclusions

The navigation and decision making architecture outlined in this paper has been found to be an effective means of controlling a UGV. Among the behaviours accomplished were:

- Autonomous patrols
- Avoidance of static obstacles (buildings, vehicles, posts, etc.)
- Avoidance of dynamic obstacles (moving people)
- Road Following
- Leader/Follower

The overall system design and architecture was found to be very effective, and especially useful in a research environment. The system modularity was very effective, allowing a number of researchers to contribute, and allowing for a sliding scale of autonomy based upon adding/removing behaviours. However the system complexity pointed out the need for software management tools, as it could be somewhat cumbersome to get all the software components started up.

The *Pure Pursuit* algorithm proved to be an extremely reliable and robust method for following paths. It worked admirably on its own, or in concert with the other components. It was also effective with poor position and sensor data, and did not become unstable when taken far from the given path.

The *Obstacle Detection* module worked well as a complement to tele-operated or semi-autonomous control, halting the Raptor UGV in dangerous situations. It was especially effective when used in concert with the *VFH* obstacle avoidance software, which was found to be extremely adept at navigating among cluttered and dynamic environments. However, it has false obstacle detections on rolling terrain, which keep it from being used for fully autonomous operations outdoors.

For the full 3D *Obstacle Avoidance* algorithm, one can conclude that slope and step hazards are sufficient metrics to determine the traversability of simple terrain such as asphalt or gravel roads. Furthermore, the ability of the UGV to effectively road-follow suggests that the Traversability Map could be used as input to a road following/recognition system. The inability of the map to accurately represent vegetation indicates that the method may not work well for off-road terrain traversal, although supplementing the Traversability Map with a terrain classification system may prove effective.

Despite the success achieved thus far, the Traversability Mapping/Obstacle Avoidance approach does not work nearly as well in heavily vegetated areas such as those encountered in off-road navigation. In these cases the algorithm is unable to differentiate between “real” obstacles such as rocks and “false” obstacles such as grass. To be successful in this environment, this approach must be supplemented with a learning algorithm which would allow the map to distinguish between various types of terrain. AISS has been conducting research in *Learned Trafficability* for a number of years and as such is well positioned to make a worthy contribution in this area.

Although not yet tested on the UGV, the *D* Lite* algorithm has been shown in simulation to be an efficient method of planning in map based environments. This will be tested on the Raptor UGV in the near future.

Finally, the *Arc Arbiter* was very useful in combining a number of varied navigation algorithms together, providing a standardized interface for these types of behaviours to interact for intelligent autonomous control.

References

[1] Broten, G., Collier, J., Giesbrect, J., Monckton, S., and Mackay, D. (2006), The Architecture for Autonomy, (DRDC Suffield TM 2006-188) Defence R&D Canada – Suffield, P.O. Box 4000, Station Main, Medicine Hat, Alberta.

[2] Giesbrecht, J. (2005), Local Navigation for Unmanned Ground Vehicles, (DRDC Suffield TM 2005-038) Defense R&D Canada – Suffield, Medicine Hat, Alberta.

[3] Giesbrecht, J. (2004), Global Path Planning for Unmanned Ground Vehicles, (DRDC Suffield TM 2004-272) Defence R&D Canada – Suffield.

[4] Simmons, R. (1996), The Curvature-Velocity Method for Local Obstacle Avoidance, *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3375–3382.

[5] Coulter, R. Craig (1992), Implementation of the Pure Pursuit Path Tracking Algorithm, (Tech Report CMU-RI-TR-92-01) Carnegie Mellon University.

[6] Amidi, O. (1990), Integrated Mobile Robot Control, Master's thesis, Carnegie Mellon University.

[7] Tipsuwan, Y. and Chow, M. (2004), Model Predictive Path Tracking Via Middleware for Networked Mobile over IP Network, In *Proceedings of the American Control Conference.*

[8] Zhang, Y., Velinsky, S., and Feng, X. (1997), On the Tracking Control of Differentially Steered Wheeled Mobile Robots, *Journal of Dynamic Systems, Measurement and Control*, 1, 455–461.

[9] Roth, S. and Batavia, P. (2002), Evaluating Path Tracker Performance for Outdoor Mobile Robot, In *Automation Technology for Off-Road Equipment.*

[10] Wit, J., Crane, C., and Armstrong, D. (2004), Autonomous Ground Vehicle Path Tracking, *Journal of Robotic Systems*, 21(8), 439–449.

[11] Khatib, M. and Chatila, R. (1995), An Extended Potential Field Approach for Mobile Robot Sensor-Based Motions, In *Proceedings of International Conference on Intelligent Autonomous Systems*, pp. 490–496.

[12] Minguez, J. and Montano, L. (2002), Robot Navigation in Very Complex, Dense and Cluttered Indoor Outdoor Environments, *Proceedings of the International Federation of Automatic Control.*

[13] Haddad, H., Khatib, M., Lacroix, S., and Chatila, R. (1998), Reactive Navigation in Outdoor Environments Using Potential Fields, In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1232–1237.

[14] Coombs, D. and Murphy, K. (2000), Driving Autonomously Offroad up to 35km/hr, In *Proceedings IEEE Intelligent Vehicles Symposium.*

[15] Kelly, A. J. (1995), Predictive Control Approach to the High-Speed Cross-Country Autonomous Navigation Problem, Ph.D. thesis, Carnegie Mellon University, Pittsburg, Pa.

[16] Shiller, Z. (1999), Motion Planning for a Mars Rover, *First Workshop on Robot Motion and Control (ROMOCO).*

[17] Cherif, M., Cherif, M., Ibanez-Guzman, J., Laugier, C., , and Goh, T. (1999), Motion Planning for an All-Terrain Autonomous Vehicle, *International Conference on Field and Service Robotics.*

[18] Dudek, G. and Jenkin, M. (2000), Computational Principles of Mobile Robotics, Cambridge, UK: Cambridge University Press. p. 10.

[19] Stentz, A. (1994), Optimal and Efficient Path Planning for Partially Known Environments, *Proceedings of IEEE International Conference on Robotics and Automation.*

[20] Murphy, Robin R. (2000), Introduction toArtificial IntelligenceRobotics, The MIT Press.

[21] Latombe, J. (1991), Robot Motion Planning, Kluwer Academic Publishers.

[22] Laumond, J. (2004), Robot Motion Planning and Control. LAAS Report 97438.

[23] Brooks, Rodney A. (1986), A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, RA-2(1), 14–23.

[24] Connell, J.H. (1990), Minimalist Mobile Robotics, Academic Press.

[25] Firby, R., Kahn, R., Prokopwicz, P., and Swain, M. (1995), An Architecture for Vision and Action, *International Joint Conference on Artificial Intelligence*, 1, 72–79.

[26] Arkin, R.C. (1987), Motor Schema Based Mobile Robot Navigation, *International Journal of Robotics Research*, 8(4).

[27] Rimon, E. and Koditschek, D.E. (1992), Exact robot navigation using artificial potential fields., *IEEE Transactions on Robotics and Automation*, 8(5), 501–518.

[28] Rosenblatt, J.K. (1995), DAMN: A Distributed Architecture for Mobile Navigation., In *Proceedings of the 1995 AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents , H. Hexmoor and D. Kortenkamp (Eds.)*, AAAI Press, Menlo Park, CA.

[29] Nilsson, Nils J. (1984), Shakey The Robot, (Technical Report 323) AI Center, SRI International, 333 Ravenswood Ave., Menlo Park, CA 94025.

[30] Meystel, A. (1988), Intelligent Control In Robotics, *Journal of Robotic Systems*, 5, 269–308.

[31] Meystel, A. (1990), Knowledge Based Nested Hierarchical Control, *Advances in Automation and Robotics*, 2, 63–152.

[32] Carbonell, J. and Veloso, M. (1988), Integrating Derivational Analogy Into A General Problem Solving Architecture, *In Proceedings of a Workshop on Case- Based Reasoning,*.

[33] Albus, J. (1991), Outline for a Theory of Intelligence, *IEEE Transactions on Systems, Man and Cybernetics*, 21(3), 473–509.

[34] Albus, J. and Meystel, A. (2001), Intelligent Systems: Architecture , Design, Control, Wiley-Interscience.

[35] Albus, J. (2001), Engineering of Mind: An Introduction to the Science of Intelligent Systems, John Wiley and Sons.

[36] Arkin, R. (1998), Behavior Based Robotics, MIT Press.

[37] Arkin, R. (1990), Integrating Behavioral, Perceptual, and World Knowledge in Reactive Navigation, *Robotics and Autonomous Systems*, 6, 105–122.

[38] Thrun, S., Bucken, A., Burgard, W., Fox, D., Frohlinghaus, T., Hennig, D., Hofmann, T., Krell, M., and Schmidt, T. (1998), Map Learning and High-Speed Navigation in RHINO, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems.*

[39] Brock, O. and Khatib, O. (1999), High-Speed Navigation Using the Global Dynamic Window Approach, In *Proceedings of the IEEE International Conference on Robotics and Automation.*

[40] Bonasso, R., Firby, J., Gat, E., Kortenkamp, D., Miller, D., and Slack, M. (1997), A Proven Three-tiered Architecture for Programming Autonomous Robots, *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2).

[41] Stentz, A. and Hebert, M. (1995), A Complete Navigation System for Goal Aquisition in Unknown Environments, In *A Complete Navigation System for Goal Acquisition in Unknown Environments.*

[42] Rosenblatt, J. and Thorpe, C. (1997), A Behavior-based Architecture for Mobile Navigation, *Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon.*

[43] Kortenkamp, D. and Bonasso, R. (1998), Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems, 1st ed, AAAi Press.

[44] Konolige, K. and Myers, K. (1998), The Saphira Architecture for Autonomous Mobile Robots, *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems.*

[45] Simmons, R. and Koenig, S. (1995), Probabalistic Navigation in Partially Observable Environments, In *Proceedings of International Joint Conference on AI*, pp. 1080–1087.

[46] Henriksen, L. and Krotkov, E. (1997), Natural Terrain Hazard Detection with a Laser Rangefinder, In *Proceedings IEEE International Conference On Robotics and Automation*, pp. 968–973, Albuquerque, New Mexico.

[47] Apostolopoulos, D. S. (2000), Technology and field demonstration of robotic search for Antarctic meteorites, *International Journal of Robotics Research*, 19(11), 1015–1032.

[48] Broten, G. and Collier, J. (2005), The Characterization of an Inexpensive Nodding Laser, (DRDC Suffield TR 2005-232) Defense R&D Canada – Suffield, Medicine Hat, Alberta.

[49] Monckton, S., Collier, J., Giesbrecht, J., Broten, G., Mackay, D., Erickson, D., Vincent, I., and Verret, S. (2006), Staged Experiments in Mobile Vehicle Autonomy II: Autonomous Land Systems Demonstration Results., (DRDC Suffield TR 2006-243) Defence R&D Canada – Suffield.

[50] Utz, H., Sablatnog, S., Enderle, S., and Kraetzschmar, G. (2002), Miro - MIddleware for Mobile Robot Applications, *IEEE Transactions on Robotics and Automation.*

[51] Broten, G., Monckton, S., Giesbrecht, J., Verret, S., Collier, J., and Digney, B. (2004), Towards Distributed Intelligence - A high level definition, (DRDC Suffield TR 2004-287) Defence R&D Canada – Suffield.

[52] Broten, G., Monckton, S., Collier, J., and Giesbrecht, J. (2006), Architecture for Autonomy, In Grant R. Gerhart, Douglas W. Gage, Charles M. Shoemaker, (Ed.), *Proceedings of SPIE, Unmanned Vehicle Research At DRDC*, Vol. 6230, p. 11, Orlando, FL.

[53] Broten, G., Monckton, S., Giesbrecht, J., and Collier, J. (2006), Software Sysetms for Robotics, An Applied Research Perspective, *International Journal of Advanced Robotic Systems*, Volume 3, 1(2005-204), 11–17.

[54] Giesbrecht, J., Collier, J., Broten, G., Verret, S., and Monckton, S. (2006), AISS Miro Manual: A Rough Guide, (DRDC Suffield TM 2006-115) Defence R&D Canada – Suffield.

[55] Monckton, S., Vincent, I., and Broten, G. (2005), A Prototype Vehicle Geometry Server: Design and development of the ModelServer CORBA Service, (DRDC Suffield TR 2005-240) Defence R&D Canada – Suffield, Medicine Hat, Alberta.

[56] Bellutta, P., Manduchi, R., Matthies, L., Owens, K., and Rankin, A. (2000), Terrain Perception for DEMO III, In *Proceedings of the 2000 Intelligent Vehicles Conference.*

[57] Goldberg, S., Maimone, M., and Matthies, L. (2002), Stereo Vision and Rover Navigation Software for Planetary Exploration, *IEEE Aerospace Conference Proceedings*.

[58] Herbert, M. and Krotkov, E. (1993), Local Perception for Mobile Robot Navigation in Natural Terrain: Two Approaches, In *Workshop on Computer Vision for Space Applications*, pp. 24–31.

[59] Kweon, S. and Kanade, T. (1992), High-Resolution Terrain Map from Multiple Sensor Data, *IEEE Transactions on Pattern Analysis and Machine Vision*, 14(2), 278–292.

[60] Lacroix, S., Mallet, A., and Bonnafous, D. (2000), Autonomous Rover Navigation on Unknown Terrains Demonstrations in the Space Museum "Cite de l'Espace" at Toulouse Automation, Albuquerque, USA, 1997, In *7th International Symp. on Experimental Robotics*, pp. 669–683, Honolulu, HI.

[61] Kelly, A. J. (1997), An Approach to Rough Terrain Autonomous Mobility, In *International Conference on Mobile Planetary Robots*, Santa Monica.

[62] Broten, G., Giesbrecht, J., and Monckton, S. (2005), World Representation Using Terrain Maps, (DRDC Suffield TR 2005-248) Defence R&D Canada – Suffield, Medicine Hat, Alberta.

[63] Collier, J, Broten, G., and Giesbrecht, J. (2006), Traversibility Analysis for Unmanned Ground Vehicles, (DRDC Suffield TM 2006-175) Defence R&D Canada – Suffield, P.O. Box 4000, Station Main, Medicine Hat, Albe.

[64] Gillespie, T.D (1992), Fundamentals of Vehicle Dynamics, Society of Automotive Engineers.

[65] Shin, D., Singh, S., and Shi, W. (1991), A Partitioned Control Scheme for Mobile Robot Path Tracking, *Proceedings of IEEE International Conference on Systems Engineering*, 1, 338–342.

[66] Murphy, K. (1992), Navigation and Retro-traverse on a Remotely Operated Vehicle, In *IEEE Sigapore International Conference on Intelligent Control and Instrumentation*.

[67] Murphy, K. (1994), Analysis of robotic vehicle steering and controller delay, In *Proceedings of 5th International Symposium on Robotics and Manufacturing*, pp. 631–636.

[68] Kelly, A. (1997), Intelligent Unmanned Ground Vehicles: Autonomous Navigation Research at Carnegie Mellon, Ch. RANGER: Feedforward Control Approach to Autonomous Navigation, pp. 105–144, Kluwer Academic Publishers.

[69] Giesbrecht, J., Mackay, D., Collier, J., and Verret, S. (2005), Path Tracking for Unmanned Ground Vehicle Navigation, (DRDC Suffield TM 2005-224) Defence R&D Canada – Suffield.

[70] Borenstein, J. and Koren, Y. (1991), The Vector Field Histogram - Fast Obstacle Avoidance for Mobile Robots, *IEEE Transactions on Robotics and Automation*, 7(3), 535–539.

[71] Gerkey, Brian, Vaughan, Richard T., and Howard, Andrew (2003), The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems, *Proceedings of the 11th International Conference on Advanced Robotics*, pp. 317–323.

[72] Ulrich, Iwan and Borenstein, Johann (1998), VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots, In *IEEE International Conference on Robotics and Automation*, pp. 1572–1577, Leuven, Belgium.

[73] Ye, Cang and Borenstein, Johann (2004), A Method for Mobile Robot Navigation on Rough Terrain, *Proc Proceedings of IEEE International Conference on Robotics and Automation*.

[74] Singh, S., Schwehr, K, Simmons, R., Smith, T., Stenz, A., Verma, V., and Yahja, A. (2000), Recent Progress In Local and Global Traversability For Planetary Rovers, *International Conference on Robotics and Automation*.

[75] Koenig, S. and Likhachev, M. (2002), D* Lite, *Proceedings of the National Conference on Artificial Intelligence*, pp. 476–483.

[76] Stentz, A. (1995), The Focussed D Algorithm for Real-Time Planning, *Proceedings of the InternationalJoint Conference on Artificial Intelligence*.

[77] Koenig, S. and Likhachev, M. (2001), Lifelong Planning A*, (Technical Report GIT-COGSCI-2002/2) Georgia Institute of Technology.

[78] Mackay, David (2005), Path Planning with D*-Lite, (DRDC Suffield TM 2005-242) Defence R&D Canada – Suffield.

[79] Monckton, S., Collier, J., Giesbrecht, J., Broten, G., Mackay, D., Erickson, D., Verret, S., and Digney, B. (2006), The ALS Project: Lessons Learned, In Gerhart, G. R., ShoeMaker, C.M., and Gage, D.M., (Eds.), *SPIE Defense and Security Symposium Unmanned Systems Technology VIII*, Vol. 6230, p. 12.

Annex A: Interface Descriptions

A.1 Sensing

A.1.1 Laser and Stereo

```
// The SICK laser returns exactly 361 data points and thus
// exactly 361 longs are allocated for each of the X,Y,Z
// coordinates and range.

const long LASER_NUM       =  361;
const long PLANES_NUM      =  4;

//! An array of  3d point groups.
typedef double Range3dLaserIDL[LASER_NUM][PLANES_NUM];

struct Range3dLaserEventIDL
{
  //! The time the scan was accquired.
  TimeIDL time;

  //! Angle of the nodding laser in radians
  double nodangle;

/* ID of the Laser */
long laser_ID;

   //! The data, X, Y, Z and the range
   Range3dLaserIDL range3d;
};
```

A.1.2 IMU

```
struct ImuIDL
{
//!Structure containing time the IMU was sampled
TimeIDL time;

/*!4 component quaternion which describes the orientation
of the IMU with respect to the fixed earth coordinate system.
The earth fixed coordinate system has X pointing North,
Y pointing East, and Z pointing down. */
double quaternion[4];

/*!Vector (X, Y and Z components) quantifying the direction
and magnitude of the instantaneously measured magnetic field
that the IMU is exposed expressed in terms of the IMU's
local coordinate system.*/
double magfield[3];

/*!Vector (X, Y and Z components) quantifying the direction
and magnitude of the instantaneously measured acceleration
expressed in terms of the IMU local coordinate system.*/
double accel[3];

/*!Vector (X, Y and Z components) quantifying the rate
of rotation of the IMU expressed in terms of the IMU
local coordinate system.*/
double angrate[3];

/*!Vector (X, Y and Z components) quantifying the roll,
pitch and yaw of the IMU expressed in terms of theroll
IMU local coordinate system.*/
double angle[3];

/*!The 3x3 orientation matrix describes the orientation
of the IMU expressed in terms of the IMU local coordinate
system.*/
double orientmatrix[3][3];
long status;
};
```

A.1.3 GPS

```
struct GpsIDL
{
  //!UTC time (seconds since the epoch)
  double utc_time;

  //Latitude and longitude (degrees).  Latitudes are positive for
  //North, negative for South.  Longitudes are positive for East,
  //negative for West.
  //!Latitude (degrees).  Latitudes are positive for north, negative
  //for south
  double latitude;

  //!Longitude (degrees). Logitudes are positive for east,
  //negative for west.
  double longitude;

  //!Altitude (meters).  Positive is above sea-level,
  //negative is below.
  double altitude;

  //!UTM easting (meters).
  double utm_e;

  //!UTM northing (meters).
  double utm_n;

  //!Horizontal dilution of precision.
  double hdop;

  //!Horizontal error (meters).
  double err_horz;

  //!Vertical error (meters).
  double err_vert;

  //!Quality of fix 0 = invalid, 1 = GPS fix, 2 = DGPS fix
  long quality;

  //!Number of satellites in view.
  long sat_count;

  //!Time since epoch seconds
  long time_sec;

  //!Time since epoch micro seconds
  long time_usec;

  //GPS Sokkia solution status
  unsigned long sol_status;

  //GPS Sokkia position type
  char pos_type[70];

  //GPS Sokkia BESTPOS mode standard deviations
  double latitudestandarddeviation;
  double longitudestandarddeviation;
  double altitudestandarddeviation;

  //GPS Sokkia BESTVEL mode velocities (m/s) (horizontal speed
  //over ground, actual direction of motion over ground and
  // vertical speed) and latency
  //in the velocity time tag (sec).
```

```
  double hor_speed;
  double direction_motion;
  double vert_speed;
  float latency;

  long status;
};
```

A.1.4 Odometry

```
//! Struct holding a point, a two dimensional coordinate.
/**
 * When our robots start flying we are really in trouble.
 *
 * This struct is easily convertible into a WorldPoint object.
 */
struct WorldPointIDL
{
  //! X coordinate in mm.
  double x;
  //! Y coordinate in mm.
  double y;
};

//! World position in (x, y, theta)
/**
 * Struct holding a point and an angle, representing the robots current
 * position.
 *
 * This struct can be converted to a WorldPosition object.
 */
struct PositionIDL
{
  //! Current robot point in world coordinates.
  WorldPointIDL point;
  //! Direction the robot points to.
  /**
   * (-PI < heading <= PI)
   */
  double heading;
};
```

A.2 Information Processing and Representation

A.2.1 Model Server

```
//This PoseMatrixIDL is a structure containing a timestamped
//4x4 homogeneous transform with respect to an external coordinate
//system. The matrix includes an upper left 3x3 rotation matrix
//and a 3x1 position vector
//on the upper left.
struct PoseTransformIDL
{
TimeIDL time;
double HTransform[4][4];
double initialUTM[3];
char poseValidFlag;
};
```

A.2.2 Terrain Map

```
enum TerrainPlanes { XMEAN, XVAR, YMEAN, YVAR, ZMEAN, ZVAR };

// A single map grid cell
typedef double MapArrayIDL[ZVAR+1][INDEX_DEPTH][INDEX_WIDTH];

struct MapArrayEventIDL
{
  //! The time the scan was accquired.
  TimeIDL time;

  //! The Pose when the was acquired
  PoseTransformIDL pose;

  //! Define the Map type
  long maptype;

   //! Depth of the map as an index = depth/gridsize.
  long index_depth;

  //! Width of the as an index = 2*width/gridsize
  long index_width;

  //! A pointer to the current starting X index in the map
  long x_curr;

  //! A pointer to the current starting Y index in the map
  long y_curr;

  //! The map .
  MapArrayIDL map;
};
```

A.2.3 Traverse Map

```
enum TraversePlanes { TRAVERSE, CONFIDENCE };

typedef double TravMapArrayIDL[CONFIDENCE+1][TINDEX_DEPTH][TINDEX_WIDTH];


struct TravMapArrayEventIDL
{
  //! The time the scan was accquired.
  TimeIDL time;

  //! The Pose when the was acquired
  PoseTransformIDL pose;

  //! Define the Map type
  long maptype;

   //! Depth of the map as an index = depth/gridsize.
  long index_depth;

  //! Width of the as an index = 2*width/gridsize
  long index_width;

  //! A pointer to the current starting X index in the map
  long x_curr;

  //! A pointer to the current starting Y index in the map
  long y_curr;

  //! The map .
  TravMapArrayIDL map;
};
```

A.2.4 Global Map

```
/*!
```

```
** A map is an MxN grid where each grid element contains sums
** and quantities that characterize the terrain.
** The constuct of a terrain map is described, in detail, in the
** Unmanned Ground Vehicle Terrain Maps Technical Report.
**
** This IDL implements a map as a [L]x[M]x[N] vector.
** The [M]x[N] 2D array represents th grid
** while the [L] index can be viewed plane associated
** with the type of data stored in the grid.
**
** Ex. Plane #1 - Mean X coordinate
**     Plane #2 - Mean Y coordinate
**     Plane #3 - Mean Z coordinate
**
*/

/* ALREADY DEFINED IN PlanPoint.idl
//! A struct ot hold a point in the planning space.
struct PlanPointIDL
{
  long x;
  long y;
};
*/

struct PlanMapEventIDL
{
  //! The time the scan for the traversability map was accquired.
  TimeIDL time;

  //! The transform from the traversability frame to the planning frame.
  double transform[4][4];

  //! The traversability map size.
  long index_depth;
  long index_width;

  //! The traversability map resolution.
  long gridsize;

  //! The traversability map.
  TravMapArrayIDL map;

  //! The starting location of the current plan segment.
  PlanPointIDL start;
  //! The goal location of the current plan segment.
  PlanPointIDL goal;

  //! The vehicle's current location in the planning space.
  PlanPointIDL vehicle;

  //! The planner's interim target location in the planning space.
  PlanPointIDL target;

  //! The planning space size.
  long depth;
  long width;
};
```

A.2.5 User Control Station

```
// A basic waypoint structure
struct WaypointIDL
{
double lat;
double lon;
double radialTolerance; // metres
// This waypoint's place in the sequence of waypoints making up the path.
long seqNumber;
};

// An array of variable length of waypoints, which describes a path
typedef sequence<WaypointIDL> WaypointListIDL;

// The structure which is sent to the vehicle
struct WaypointGroupIDL
{
WaypointListIDL PointList;
};
```

In addition to these structures, an interface was described which allows this path to be sent to the vehicle. In addition to allowing the control station to retrieve information about the vehicle's status, it also allows it to set the vehicle's Patrol Mode, causing it to execute the path repeatedly.

```
interface VehiclePlan
{
// Command the vehicle to follow a path
WaypointGroupIDL setWaypointList(in WaypointGroupIDL waypoints);

// Query what path the vehicle is currently on
WaypointGroupIDL getWaypointList();

// Query what waypoint of the path the vehicle is working on
WaypointIDL getCurrentWaypoint();

// True makes the vehicle will loop through its waypoints continuously
boolean setPatrolMode(in boolean patrolmode);

// Query the patrol mode state
boolean getPatrolMode();
};
```

This page intentionally left blank.

Annex B: Module Parameters

These tables show the configurable parameters for each algorithm described in Section 5.

B.1 Obstacle Detection Parameters

Parameter	Default	Description
SafetyHeight	0.3	The maximum safe obstacle height(m)
SafetyDepth	0.3	The maximum safe obstacle depth(m)
SafetyDist	3	Distance(m) in front of the vehicle to check for obstacles
VehicleWidth	1	The approximate width of the vehicle(m)
DefaultYMax	3	Distance(m) to the left/right of the vehicle to check for obstacles
LaserHeight	2	The height(m) of the LRF above the ground
NumHitsTolerance	5	Number of LRF returns outside the safety height/depth required to trigger an obstacle detection
MaxLaserRange	8191	Maximum LRF range(mm)
CheckPosObs	true	Whether or not to look for positive obstacles
CheckNegObs	true	Whether or not to look for negative obstacles
ResetCounter	200	Number of laser scans to wait after a halt before rechecking for obstacles

Table B.1: *Obstacle Detection Component's Configuration Parameters*

B.2 Vector Field Histogram Parameters

Parameter	Default	Description
CellSize	0.1	The size of each grid cell in the map (meters). will search.
WindowDiameter	121	Dimension of the occupancy grid (number of cells in diameter).
SectorAngle	5	Histogram angular resolution (degrees).
SafetyDist1ms	1.0	The minimum distance the robot can get to an obstacle when moving at 1m/s (meters).
MaxSpeed	3	The maximum allowable speed of the robot (m/s).
MaxSpeedNarrowOpening	1	The maximum allowable speed of the robot through a narrow opening(m/s).
MaxSpeedWideOpening	2	The maximum allowable speed of the robot through a wide opening(m/s).
MaxAcceleration	0.5	The maximum allowable acceleration of the robot (m/s^2).
MinTurnrate	5	The minimum allowable turnrate of the robot (deg/s).
MaxTurnrate1ms	40	The maximum allowable turnrate of the robot when moving at 1m/s (deg/s).
FreeSpaceCutoff1ms	20000	The higher the value, the closer the robot will get to obstacles before avoiding while moving at 1 m/s (unitless).
ObsCutoff1ms	20000	The higher the value, the closer the robot will get to obstacles before avoiding while moving at 1 m/s (unitless).
WeightDesiredDir	7.0	Bias for the robot to turn to move toward goal position.
WeightCurrDir	3.0	Bias for the robot to continue moving in current direction of travel.
DistanceEpsilon	3.0	Planar distance from the target position that will be considered acceptable (meters).
AngleEpsilon	10	Angular difference from target angle that will considered acceptable (degrees).
MaxRobotRadius	1.0	The largest distance from the center of the robot to its outside (meters).

Table B.2: VFH *Component's Configuration Parameters*

B.3 Obstacle Avoidance Parameters

Parameter	Default	Description
CurveMakingConst	0.01	A distance(m) which the algorithm steps through to construct an approximation of the candidate arc
VehicleRadius	0.5	A radius(m) of a circle which approximates the size of the vehicle
MaxVelocity	2.0	The top speed(m/s) which the algorithm will allow for any arc
MinVelocity	0.5	The lowest speed(m/s) the algorithm will suggest for any arc before vetoing it
NearZone	0	The number of map rows immediately in front of the vehicle to ignore obstacles. Makes it possible to ignore parts of the map.
FarZone	39	The number of map rows immediately in front of the vehicle to evaluate. Makes it possible to ignore parts of the map which are too far away to be important.
DistFactor	0	A percentage by which to discount the cells estimation for distance
MinDistDiscount	1	A number between 0 and 1 which indicates how low we can discount for distance (1 = no discount)
MapDepth	16000	The depth of the Traversability Map(mm).
MapWidth	16000	The width of the Traversability Map(mm).
MapGridsize	0.01	The size of each cell in the Traversability Map(mm).
DiscountSpeeds	false	Whether or not the vehicles speed will be controlled based on the obstacle map.
CostUnknownCells	false	Whether or not to consider unknown portions of the map as obstacles.
VetoUnknownCells	false	Whether or not to consider unknown portions of the map as obstacles.

Table B.3: *Obstacle Avoidance Component's Configuration Parameters*

B.4 D* Lite Parameters

Parameter	Default	Description
MaxLookahead	20.0	The maximum distance (in planning grid units) along the planned path from the point on the path closest to the vehicle's current position that the tracking algorithm will search.
MinLookahead	7.0	The minimum distance (in planning grid units) along the planned path from the point on the path closest to the vehicle's current position that the tracking algorithm will search.
TrackingTolerance	30.0	The maximum distance (in planning grid units) the vehicle can be from the planned path without forcing a replanning episode.
PlanUpdateDelayTime	5.0	The delay (s) prior to the first call to generate a ArcVoteIDL event.
PlanEventInterval	1.0	The interval (s) between successive ArcVoteIDL events.

Table B.4: D* Lite *Component's Configuration Parameters.*

B.5 Arc Arbiter Parameters

Parameter	Default	Description
ObsWeight	1	The importance of Obstacle Avoidance's votes (1 being standard)
PursuitWeight	1	The importance of Pure Pursuit's votes
PlannerWeight	1	The importance of Find Path's votes
LaserSafetyWeight	1	The importance of Obtacle Detections's votes
TransVelocitySetPoint	0.5	The maximum allowed vehicle speed (m/s)
StaleVoteTimeSec	1	How long to hang on to a vote before considering it stale
StaleVoteTimeUSec	500000	How long to hang on to a vote before considering it stale

Table B.5: *Arc Arbiter Component's Configuration Parameters*

Distribution list

DRDC Suffield TR 2007-300

Internal distribution

DRDC - Suffield

1	DG/DDG/Ch. Sci. /SMO
1	H/AISS
3	Lead Author
4	Other Author (1 each)
2	Library - 1 hardcopy/1 softcopy

Other DRDC

1	DRDKIM-softcopy
1	DRDC - Valcartier, Benoit Ricard
1	DSTL
1	DSTL-6

Total internal copies: 15

External distribution

NDHQ

1	DGSP
1	CFEC
1	DDA
1	DLSC
1	DAD
1	DLA

Total external copies: 6

Total copies: 21

This page intentionally left blank.

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence R&D Canada – Suffield PO Box 4000, Station Main, Medicine Hat, AB, Canada T1A 8K6	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

A Navigation and Decision Making Architecture for Unmanned Ground Vehicles

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Giesbrecht, J.; Collier, J.; Broten, G.; Monckton, S.; Mackay, D.

5. DATE OF PUBLICATION (Month and year of publication of document.) December 2007	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 98	6b. NO. OF REFS (Total cited in document.) 79

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Report

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Suffield
PO Box 4000, Station Main, Medicine Hat, AB, Canada T1A 8K6

9a. PROJECT NO. (The applicable research and development project number under which the document was written. Please specify whether project or grant.)	9b. GRANT OR CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Suffield TR 2007-300	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
(   ) Defence departments and defence contractors; further distribution only as approved
(   ) Defence departments and Canadian defence contractors; further distribution only as approved
(   ) Government departments and agencies; further distribution only as approved
(   ) Defence departments; further distribution only as approved
(   ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

Unlimited

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

Researchers at Defence R&D Canada – Suffield, under the Autonomous Land Systems (ALS) and Cohort projects, have been working to extend/enhance the capabilities of Unmanned Ground Vehicles (UGVs) beyond tele-operation. The goal is to create robotic platforms that are effective with minimal human supervision in outdoor environments. This report is a summary of the progress made in high level vehicle control, specifically the implementation and testing of algorithms providing point-to-point navigation and decision making capabilities for UGVs. To reach goals by traversing unknown terrain requires a number of navigation functions, including path tracking, obstacle avoidance, path planning and decision making modules. This report presents details of the theoretical underpinnings, the software design architecture, and results of implementing autonomous navigation and decision making software on a robotic platform, given competing priorities and limited sensing technologies.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

robotics
navigation
obstacle avoidance
autonomy